

**TITLE OF THE INVENTION**

METHOD AND SYSTEM FOR ON-SCREEN ANIMATION OF  
DIGITAL OBJECTS OR CHARACTERS

**FIELD OF THE INVENTION**

[0001] The present invention relates to the digital entertainment industry and to computer simulation. More specifically, the present invention concerns a method and system for on-screen animation of digital objects or characters.

**BACKGROUND OF THE INVENTION**

[0002] It's the nature of the digital entertainment industry to continuously push the boundaries of creativity. This drive is very strong in the fields of three-dimensional (3D) animation, visual effects and gaming. Hand animation and particle systems are reaching their natural limits.

[0003] Procedural animation, which is driven by artificial intelligence (AI) technique is the new frontier. AI animation allows augmenting the abilities of digital entertainers across disciplines. It gives game designers the breadth, independence and tactics of film actors. Film-makers get the depth and programmability of an infinite number and real time game style characters.

[0004] Until recently, the field of AI animation was limited to a handful of elite studios with a large development team that developed their own expensive proprietary tools. This situation is akin to the case of early filmmakers such as the Lumière Brothers, who had no choice but to build their own cameras.

**[0005]** For over twenty years, the visual effects departments of film studios have increasingly relied on computer graphics for whenever a visual effect is too expensive, too dangerous or just impossible to create any other way than via a computer. Unsurprisingly, the demands on an animator's artistic talent to produce even more stunning and realistic visual effects have also increased. Nowadays, it is not uncommon that the computer animation team is just as important to the success of a film as the lead actors.

**[0006]** Large crowd scenes, in particular battle scenes, are ideal candidates for computer graphics techniques since the sheer number of extras required make them extremely expensive, their violent nature make them very dangerous, and the use of fantastic elements such as beast warriors make them impractical, if not impossible, to film with human extras. Given the complexity, expense, and danger of such scenes, it is clear that an effective artificial intelligence (AI) animation solution is preferable to actually staging and filming such battles with real human actors. However, despite the clear need for a practical commercial method to generate digital crowd scenes, a satisfactory solution has been a long time in coming.

**[0007]** Commercial animation packages such as Maya™ by Alias Systems have made great progress in the last twenty years to the point that virtually all 3D production studios rely on them to form the basis of their production pipelines. These packages are excellent for producing special effects and individual characters. However, crowd animation remains a significant problem.

**[0008]** According to traditional commercial 3D animation techniques, animators must laboriously keyframe the position and orientation of each character frame by frame. In addition to requiring a great deal of the animator's time, it also requires expert knowledge on how intelligent characters actually interact. When the number of characters to be animated is more than a handful,

this task becomes extremely complex. Animating one fish by hand is easy; animating fifty (50) fish by hand can become time consuming.

**[0009]** Non-linear animation techniques such as Maya's Trax Editor™, try to reduce the workload by allowing the animator to recycle clips of animations in a way that is analogous to how sound clips are used. According to this clip recycling technique, an animator must position, scale, and composite each clip. Therefore, to make a fish swim across a tank and turn to avoid a rock, the animator repeats and scales the swim clip and then adds a turn clip. Although this reduces the workload per character, it still must be repeated for each individual character, e.g. the fifty (50) fish.

**[0010]** Rule-based techniques present a more practical alternative to their laborious keyframe counterparts. Particle systems try to reduce the animator's burden by controlling the position and orientation of the character via simple rules. This is effective for basic effects such as a school of fish swimming in a straight-line. However the characters do not avoid each other and they all maintain the exact same speed. Moreover, animation clip control is limited to simple cycling. For example, it is very difficult to get a shark to chase fish and the fish to swim away, let alone for the shark to eat the fish and have them disappear.

**[0011]** A solution to this problem is to develop an AI solution in-house. Writing proprietary software may present the animator with the ability to create a package specifically designed for a given project, but it is often an expensive and risky proposition. Even if the necessary expertise can be found, it is most often not in the company's best interest to spend time and money on a non-core competency. In the vast majority of cases, the advantages of buying a proven technology outweigh this expensive, high-risk alternative.

**[0012]** In the computer game field, game AI has been in existence since the dawn of video games in the 1970s. However, it has come a long way since the creation of Pong™ and Pac-Man™. Nowadays, game AI is increasingly becoming a critical factor to a game's success and game developers are demanding more and more from their AI. Today's AI need to be able to seemingly think for themselves and act according to their environment and their experience giving the impression of intelligent behaviour, i.e. they need to be autonomous.

**[0013]** Game AI makes games more immersive. Typically game AI is used in the following situations:

- to create intelligent non-player characters (NPCs), which could be friend or foe to the player-control characters;
- to add realism to the world. Simply adding some non-essential game AI that reacts to the changing game world can increase realism and enhance the game experience. For example, AI can be used to fill sporting arenas with animated spectators or to add a flock of bats to a dungeon scene;
- to create opponents when there are none. Many games are designed for two or more players however, if there is no one to play against intelligent AI opponents are needed; or
- to create team members when there are not enough. Some games require team play, and game AI can fill the gap when there are not enough players.

**[0014]** Typically, in a conventional computer game, the main loop contains successive calls to the various layers of the virtual world, which could include the game logic, AI, physics, and rendering layers. The game logic layer determines the state of the agent's virtual world and passes this information to the AI layer. The AI layer then decides how the agent reacts according to the agent's characteristics and its surrounding environment. These directions are then sent to the physics layer, which enforces the world's physical laws on the game objects. Finally, the rendering layer uses data sent from the physics layer to produce the onscreen view of the world.

### **OBJECTS OF THE INVENTION**

**[0015]** An object of the present invention is therefore to provide an improved method and system for on-screen animation of digital entities.

### **SUMMARY OF THE INVENTION**

**[0016]** A method and system for on-screen animation of digital entities according to the present invention allows controlling the interaction of image entities within a virtual world. Some of the digital entities are defined as autonomous image entities (AIE) that can represent characters, objects, virtual cameras, etc, that behave in a seemingly intelligent and autonomous way. The virtual world includes autonomous and non-autonomous entities that can be graphically represented on-screen in addition to other digital representation which can or cannot be represented graphically on a computer screen or on another display.

**[0017]** Generally stated, the method and system allows generating seemingly intelligent image entities motion with the following properties:

**[0018]** 1) Intelligent Navigation

**[0019]** Intelligent navigation in a world is handled on two conceptual levels. The first level is purely reactive and it includes autonomous image entities (AIE) attempting to move away from intervening obstacles and barriers as they are detected. This is analogous to the operation of human instinct in reflexively pulling one's hand away from a hot stove.

**[0020]** The second level involves forethought and planning and is analogous to a person's ability to read a subway map in order to figure out how to get from one end of town to the other.

**[0021]** Convincing character navigation is achieved by combining both levels. Doing so enables a character to navigate paths through complex maps while at the same time being able to react to dynamic obstacles encountered in the journey.

**[0022]** 2) Intelligent Animation Control

**[0023]** In addition to being seemingly intelligently moved within the virtual world, AIEs' animations can be driven based on their stimuli.

**[0024]** The simplest level of animation control allows to, for example, play back an animation cycle based on the speed of motion of a character's travel. For example, a character's walk animation can be scaled according to the speed of its movement.

**[0025]** On a more complex level, AIEs can have multiple states and multiple animations associated with those states, as well as possible special-case transition animations when moving from state to state. For example, a character can seamlessly run, slow down as it approaches a target, blending through a walk cycle and eventually ending up at, for example, a "talk" cycle.

The resulting effect is a character that runs towards another character, slows down and starts talking to them.

**[0026]**            3) Interactivity

**[0027]**            By specifying reactive-level and planning-level, AIEs can adapt to a changing environment.

**[0028]**            A method and system for on-screen animation of digital entities according to the present invention allows defining an AIE that is able to navigate a world while avoiding obstacles, dynamic or otherwise. Adding more obstacles or changing the world can be achieved in the virtual world representation, allowing characters to understand their environment and continue to be able to act appropriately within it.

**[0029]**            It is to be noted that the expression “virtual world” and “digital world” are interchangeable herein.

**[0030]**            AIEs’ brains can also be described with complex logic via a subsystem referred to herein as “Action Selection”. Using sensors to read information about the virtual world, decision trees to understand that information, and commands to execute resulting actions, AIEs can accomplish complex tasks within the virtual world, such as engaging in combat with enemy forces.

**[0031]**            A system for on-screen animation of digital entities according to the present invention may include:

**[0032]**            A) A solver

**[0033]** The system includes an Autonomous Image Entity Engine (AIEE). The engine calculates and updates the position and orientation of each character for each frame, chooses the correct set of animation cycles, and enables the correct simulation logic. Within the Autonomous Entity Engine is the solver, which allows the creation of intelligent entities that can self-navigate in the geometric world. The solver drives the AIEs and is the container for managing these AIEs and other objects in the virtual world.

**[0034]** B) Autonomous and non-autonomous image entities, including groups of image entities

**[0035]** Image entities come in two forms: autonomous and non-autonomous. In simple terms, an autonomous image entity (AIE) acts as if it has a brain and is controlled by in a manner defined by attributes it has been assigned. The solver controls the interaction of these autonomous image entities with other entities and objects within the world. Given this very general specification, an AIE can control anything from a shape-animated fish, a skeletal-animated warrior, or a camera. Once an AIE is defined it is assigned characteristics, or attributes, which define certain basic constraints about how the AIE is animated.

**[0036]** A non-autonomous image entity does not have a brain and must be manipulated by an external agent. Non-autonomous image entities are objects in the virtual world that, even though they may potentially interact with the world, are not driven by the solver. They can include objects such as player-controlled characters, falling rocks, and various obstacles.

**[0037]** Once an AIE is defined, characteristics or attributes, which define certain basic constraints about how the AIE can move, are assigned thereto. Attributes include, for example, the AIE's initial position and orientation, its maximum and minimum speed and acceleration, how quickly it can turn, and



if the AIE hugs a given surface. These constraints will be obeyed when the AIE's position and orientation are calculated by the solver. The AIE can then be assigned pertinent behaviours to control its low-level locomotive actions. Behaviours generate steering forces that can change an AIE's direction and/or speed for example. Without an active behaviour, an AIE would remain moving in a straight line at a constant speed until it collided with another object.

**[0038]** Non-autonomous characters are objects in the digital world that, even though they may potentially interact with the world, are not driven by the solver. These can range from traditionally animated characters (e.g. the leader of a group) to objects (e.g. boulders and trees) driven by a dynamic solver. The method and system according to the present invention allows interaction among characters. For example, a group of autonomous characters could follow a non-autonomous leader character animated by traditional means, or the group could run away from a physics-driven boulder.

**[0039]** C) Paths and waypoint networks

**[0040]** Paths and waypoint networks are used to guide an AIE within the virtual world.

**[0041]** A path is a fixed sequence of waypoints that AIEs can follow. Each waypoint can be assigned speed limits to control how the AIE approaches it (e.g. approach this waypoint at this speed). Paths can be used to build racetracks, attack routes, flight paths, etc.

**[0042]** A waypoint network allows defining the "navigable" space in world, clearly defining to AIEs what possible routes they can take in order to travel from point to point in the world.

**[0043]** D) Behaviours

**[0044]** Behaviours provide AIEs with reactive-level properties that describe their interactions with the world. An AIE may have any number of behaviours that provide it with such instincts as avoiding obstacles and barriers, seeking to or fleeing from other characters, “flocking” with other characters as group, or simply wandering around.

**[0045]** Behaviours allow producing “desired motion” and desires from multiple behaviours can be combined to produce a single desired motion for the AIE to follow. Behaviour intensities (allowing scaling up or down of a behaviour’s produced desired motion), behaviour priorities (allowing higher priority behaviours to completely override the effect of lower priority ones), and behaviour blending (allowing a behaviour’s desired motion to be “fade in” and “fade out” over time), can be used to control the relative effects of different behaviours.

**[0046]** E) Action Selection

**[0047]** Action Selection allows enabling AIEs to make decisions based on information about their surrounding environment. As Behaviours can be thought of as instincts, Action Selection can be thought of as higher-level reasoning, or logic.

**[0048]** Action Selection is fuelled by “sensors” that allow AIEs to detect various kinds of information about the world or about other AIEs.

**[0049]** Results of sensors’ detections are saved into “datum” and this data can be used to drive binary decision trees, which provide the “if..then” logic defining a character’s high-level actions.

**[0050]** Finally, obeying a decision tree causes the character to make a decision, which is basically a group of commands. These commands provide

the character with the ability to modify its behaviours, drive animation cycles, or update its internal memory.

**[0051]** F) Animation Control

**[0052]** Another feature of a method for on-screen animation of digital entities according to the present invention is its ability to control an AIE's animations based on events in the world. By defining animation cycles and transitions between animations, the method can be used to efficiently create a seamless, continuous blend of realistic AI-driven character animation.

**[0053]** More specifically, in accordance with a first aspect of the present invention, there is provided a method for on-screen animation of digital entities comprising:

**[0054]** providing a digital world including image object elements;

**[0055]** providing at least one autonomous image entity (AIE); each the AIE being associated with at least one AIE animation clip, and being characterized by a) attributes defining the at least one AIE relatively to the image objects elements, and b) at least one behaviour for modifying at least one of the attributes; the at least one AIE including at least one virtual sensor for gathering data information about at least one of the image object elements or other one of the at least one AIE;

**[0056]** initializing the attributes and selecting one of the behaviours for each of the at least one AIE;

**[0057]** for each the at least one AIE:

using the at least one sensor to gather data information about at least one of the image object elements or other one of the at least one AIE; and

using a decision tree for processing the data information resulting in at least one of i) triggering one of the at least one AIE animation clip according to the attributes and selected one of the at least one behaviour, and ii) selecting one of the at least one behaviour.

**[0058]** According to a second aspect of the present invention, there is provided a system for on-screen animation of digital entities comprising:

**[0059]** an art package to create a digital world including image object elements and at least one autonomous image entity (AIE) and to create AIE animation clips; and

**[0060]** an artificial intelligence agent to associate to an AIE a) attributes defining the AIE relatively to the image objects elements, b) a behaviour for modifying at least one of the attributes, c) at least one virtual sensor for gathering data information about at least one of the image object elements or other AIEs, and d) an AIE animation clips; the artificial intelligence agent including an autonomous image entity engine (AIEE) for updating each AIE's attributes and for triggering for each AIE at least one of a current behaviour and one of the at least one animation clip based on the current behaviour and the data information gathered by the at least one sensor.

**[0061]** According to a third aspect of the present invention, there is provided a system for on-screen animation of digital entities comprising:

**[0062]** means for providing a digital world including image object elements;

**[0063]** means for providing at least one autonomous image entity (AIE); each the AIE being associated with at least one AIE animation clip, and being characterized by a) attributes defining the at least one AIE relatively to the image objects elements, and b) at least one behaviour for modifying at least one of the attributes; the at least one AIE including at least one virtual sensor for gathering data information about at least one of the image object elements or other one of the at least one AIE;

**[0064]** means for initializing the attributes and selecting one of the behaviours for each of the at least one AIE;

**[0065]** means for using the at least one sensor to gather data information about at least one of the image object elements or other one of the each the at least one AIE;

**[0066]** means for using a decision tree for processing the data information;

**[0067]** means for triggering one of the at least one AIE animation clip according to the attributes and selected one of the at least one behaviour; and

**[0068]** means for selecting one of the at least one behaviour.

**[0069]** A method and system for animating digital entities according to the present invention can be used in applications where there is a need for seemingly reaction of characters and objects, for example:

**[0070]** – In the special effects field for controlling or pre-visualizing, for example, motion of hundreds of rats running down a street or 10,000 soldiers fighting hand-to-hand;

**[0071]** – in game development;

**[0072]** – in 3D animation;

**[0073]** – in cinematics (cut scenes);

**[0074]** – in training systems. For example, the present invention would help implementing better automobile car driving training system with intelligent automobiles and pedestrians.

**[0075]** The method and system from the present invention provides software modules to create and control intelligent characters that can act and react to their worlds, such as:

**[0076]** - intelligent navigation. Using dynamic path finding and collision avoidance, AI characters can smoothly move from point A to point B and avoid running into anything in their way;

**[0077]** - intelligent animation control. Using animation blending, AI characters look natural by correctly choosing the correct animations, scales, and blends; and

**[0078]** - interactivity. Using sophisticated sensor and decision-making systems, AI characters can learn about their worlds and respond to them accordingly from stopping at a stop sign to hunting down escaped prisoners;

**[0079]** The method and system for on-screen animation of digital entities provides the following advantages:

**[0080]** - provide the means to create seemingly intelligent and sophisticated AI characters that act and react according to their changing environment;

**[0081]** - provide sophisticated artificial intelligence techniques that may be too specialized or costly to develop in-house;

**[0082]** - give the ability (time and tools) to refine content, which is all about fine-tuning. The method and system according to the present invention helps fine-tuning animation by allowing the AI to be up and running faster, and by providing real-time feed-back tools. Using a method and system for on-screen animation of digital entities from the present invention helps animators and designers to become independent from programmers.

**[0083]** Other objects, advantages and features of the present invention will become more apparent upon reading of the following non-restrictive description of preferred embodiments thereof, given by way of example only with reference to the accompanying drawings.

#### **BRIEF DESCRIPTION OF THE DRAWINGS**

**[0084]** In the appended drawings:

**[0085]** Figure 1 is a flowchart illustrating a method for on-screen animation of digital entities according to an illustrative embodiment of a first aspect the present invention;

**[0086]** Figure 2 is a schematic view illustrating a two-dimensional barrier to be used with the method of Figure 1;

**[0087]** Figure 3 is a schematic view illustrating a three-dimensional barrier to be used with the method of Figure 1;

**[0088]** Figure 4 is a schematic view illustrating a co-ordinate system used with the method of Figure 1;

**[0089]** Figure 5 is a flowchart illustrating step 110 from Figure 1 corresponding to the use of a decision tree to issue commands;

**[0090]** Figure 6 is a bloc diagram of a system for on-screen animation of digital entities according to a first illustrative embodiment of a second aspect of the present invention;

**[0091]** Figure 7 is a still image taken from a first example of animation created using the method from Figure 1 and related to a representation of a school of fish and of a shark in a large aquarium;

**[0092]** Figure 8 is a flowchart of a behaviour decision tree used in action selection for the animation illustrated in Figure 7;

**[0093]** Figure 9 is a behaviour decision tree to determine the behaviour of roman soldier in a second example of animation created using the method of Figure 1 and related to a representation of a battle scene between roman soldiers and beast warriors;

**[0094]** Figure 10 is a behaviour decision tree to determine the behaviour of beast warriors in the second example of animation created using the method of Figure 1;

**[0095]** Figures 11A-11D are still images of a bird's eye view of the battle field from the second example of animation created using the method of



Figure 1, illustrating the march of the roman soldiers and beast warriors towards one-another;

**[0096]** Figure 12 is a decision tree allowing to select the animation clip to trigger when a beast warrior and a roman soldier engage in battle in the second example of animation created using the method of Figure 1;

**[0097]** Figures 13A-13C and 14 are still images taken from on-screen animation of the battle scene according to the second example of animation created using the method of Figure 1; and

**[0098]** Figures 15 and 16 are bloc diagrams illustrating a system for on-screen animation of digital entities according to a second illustrative embodiment of a first aspect of the present invention.

## **DETAILED DESCRIPTION OF THE INVENTION**

**[0099]** A method 100 for on-screen animation of digital entities according to an illustrative embodiment of a first aspect of the invention will now be described, with reference first to Figure 1 of the appended drawings.

**[00100]** The method 100 comprises the following steps:

**[00101]** 102 – providing a digital world including image object elements;

**[00102]** 104 – providing autonomous image entity (AIE), associated with corresponding animation clips;

**[00103]** 106 – defining and initializing the attributes and behaviours for each AIE;

**[00104]**            108 – AIEs using sensors to gather data information about image object elements or other AIEs; and

**[00105]**            110 – AIEs processing the data information using decision trees, resulting in either:

**[00106]**            112 – each AIE triggering a behaviour; or

**[00107]**            114 – each AIE triggering an animation.

**[00108]**            These general steps will now be further described.

**[00109]**            A digital world model including image object elements is first provided in step 102. The image object elements include two or three-dimensional (2D or 3D) graphical representations of objects, autonomous and non-autonomous characters, building, animals, trees, etc. It also includes barriers, terrains, and surfaces. The concepts of autonomous and non-autonomous characters and objects will be described hereinabove in more detail.

**[00110]**            As it is believed to be commonly known in the art, the graphical representation of objects and characters can be displayed, animated or not, on a computer screen or on another display device, but can also inhabit and interact in the virtual world without being displayed on the display device.

**[00111]**            Barriers are triangular planes that can be used to build walls, moving doors, tunnels, etc. Terrains are 2D height-fields to which AIE can be automatically bound (e.g. keep soldier characters marching over a hill). Surfaces are triangular planes that may be combined to form fully 3D shapes to which autonomous characters can also be constrained.

**[00112]** In combination, these elements are to be used to describe the world in which the characters inhabit.

**[00113]** In addition to the image object elements, the digital world model includes a solver, which allows managing autonomous image entities (AIE), including autonomous characters, and other objects in the world.

**[00114]** The solver can have a 3D configuration, to provide the AIE with complete freedom of movement, or a 2D configuration, which is more computationally efficient, and allows an animator to insert a greater number of AIE in a scene without affecting performance of the animation system.

**[00115]** A 2D solver is computationally more efficient than a 3D solver since the solver does not consider the vertical (y) co-ordinate of an image object element or of an AIE. The choice between the 2D and 3D configuration depends on the movements that are allowed in the virtual world by the AIE and other objects. If they do not move in the vertical plane then there is no requirement to solve for in 3D and a 2D solver can be used. However, if the AIE requires complete freedom of movement, a 3D solver is used. It is to be noted that the choice of a 2D solver does not limit the dimensions of the virtual world, which may be 2D or 3D. The method 100 provides for the automatic creation of a 2D solver with default settings whenever an object or an AIE is created before a solver.

**[00116]** The following table shows examples of parameters that can be used to define the solver:

Parameter	Description
<i>Type</i>	Can be either 2D or 3D.

Parameter	Description
<i>Start Time</i>	The start time of the solver. When the current time in the system embedding the solver is less than the Start Time, the solver does not update any AIE.
<i>Width</i>	The size of the world in the z direction. The width, depth, and height form the bounding box of the solver. Only the AIEs whose within $-Width/2$ and $Width/2$ from the solver centre are updated. The solver will not update AIEs outside this range.
<i>Depth</i>	The size of the world in the x direction. The width, depth, and height form the bounding box of the solver. Only the AIEs whose within $-Depth/2$ and $Depth/2$ from the solver centre are updated. The solver does not update AIEs outside this range.
<i>Height</i>	The size of the world in the y direction. The width, depth, and height form the bounding box of the solver. Only the AIEs whose within $-Height/2$ and $Height/2$ from the solver centre are updated. The solver will not update AIEs outside this range.
<i>Grid Type</i>	Can be either 2D or 3D. The grid is a space-partitioning grid used internally by the system to optimize the search for barriers. Increasing the number of partitions in the grid generally decreases the computational time needed to update the world, but increases the solver memory usage. A 2D grid can be used in a 3D world and is equivalent to using a 3D grid with 1 height partition. Such a parameter is relevant only when barriers are defined in the world as will be explained hereinbelow in more detail.
<i>Grid Width Partitions</i>	The number of partitions in the space-partitioning grid along the z-axis. This parameter is relevant only if barriers are defined in the world. The value is set greater than or equal to 1 and is also a power of 2, <i>i.e.</i> 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc.
<i>Grid Depth Partitions</i>	The number of partitions in the space-partitioning grid along the x-axis. This parameter is relevant only if barriers are defined in the world. The value is set greater than or equal to 1 and is also a power of 2, <i>i.e.</i> 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc.
<i>Grid Height Partitions</i>	The number of partitions in the space-partitioning grid along the y-axis. This parameter is relevant only if barriers are defined in the world. The value is set greater than or equal to 1 and is also a power of 2, <i>i.e.</i> 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, etc.

Parameter	Description
<i>Use Cache</i>	This parameter allows to define whether or not a cache will be used. If a cache is activated, each frame that is computed is cached. When the animation platform requests the locations, orientations, and speeds of characters for a certain frame, the solver first searches for the information in the cache. If the solver does not find the required information, it calculates it. When a scene is saved, the cache is saved to the cache file. When a scene is loaded, the cache is loaded from the cache file.
<i>Center Position</i>	The centre of the solver's bounding box given as x, y, z co-ordinates.
<i>Random Seed</i>	The random seed allows generating a sequence of pseudo-random numbers. The same seed will result in the same sequence of generated random numbers. Random numbers are used for wander behaviours, behaviours with probabilities, and random sensors as will be explained hereinbelow. For example, if an AIE has a Wander Around behaviour, using the same seed, the Wander Around behaviour will produce the same random motions each time the scene is played and the AIE will move in the exact same way each time if no AIE are added to the scene. By changing the random seed, the Wander Around behaviour will generate a new sequence of random motions and the character will move differently than before.

**[00117]** Non-autonomous characters are objects in the digital world that, even though they may potentially interact with the digital world, are not driven by the solver. These can range from traditionally animated characters (e.g. the leader of a group) to objects (e.g. boulders and trees) driven by the solver.

**[00118]** Barriers are equivalent to one-way walls, i.e. an object or an AIE inhabiting the digital world can pass through them in one direction but not in the other. When a barrier is created, spikes (forward orientation vectors) are used to indicate the side of the wall that can be detected by an object or an

AIE. Therefore, an object or an AIE can pass from the non-spiked side to the spiked side, but not vice-versa. It is to be noted that a specific behaviour must be defined and activated for an AIE to attempt to avoid the barriers in the digital world (Avoid Barriers behaviour). The concept of behaviours will be described hereinbelow in more detail.

**[00119]** As illustrated in Figures 2 and 3 respectively, a barrier is represented in a 2D solver by a line and by a triangle in a 3D solver. The direction of the spike for 2D and 3D barriers is also shown in Figures 2-3 (see arrows 10 and 12 respectively) where P1-P3 refers to the order in which the points of the barrier are drawn. Since barriers are unidirectional, two-sided barriers are made by superimposing two barriers and by setting their spikes opposite to each other.

**[00120]** Each barrier can be defined by the following parameters:

Parameter	Description
<i>Exists</i>	This parameter allows the system to determine whether or not the barrier exists in the solver world. If this is set to <i>off</i> the solver ignores the barrier.
<i>Collidable</i>	This parameter allows the system to determine whether or not collisions with other collidable objects will be resolved. If this parameter is set to <i>off</i> characters can pass through the barrier from either side.
<i>Opaque</i>	This parameter allows set whether or not objects can see through the barrier using a sensor as will be explained hereinbelow.
<i>Surface</i>	This parameter allows set whether or not the barrier will be considered as a surface. A barrier that is a surface is considered for surface hugging by the solver.

Parameter	Description
<i>Use Bounding Box</i>	This parameter allows the system to determine whether or not to create barriers based on the bounding boxes for the selected objects. If the currently active solver has a 2D configuration then the barriers created with this option will only be created around the bounding-perimeter. If the solver is 3D, then barriers will be created and positioned the same way as the bounding box for the object.
<i>Use Bounding Box Per Object</i>	If the "Use Bounding Box "parameter is enabled and this option is also enabled a barrier-bounding box per selected object will be created. If it is disabled, a barrier-bounding box will be created at the bounding box for the group of selected items.
<i>Reverse Barrier Normal</i>	This parameter reverses the normals for the selected barriers.
<i>Group Barriers</i>	When this parameter is activated, all barriers are grouped under a group node.

**[00121]** As it is commonly known, a bounding box is a rectilinear box that encapsulates and bounds a 3D object.

**[00122]** When barriers are defined in the world, the space-partitioning grid in the AI solver may be specified in order to optimize the solver calculations that concern barriers.

**[00123]** The space-partitioning grid allows to optimize the computational time needed for solving, including updating each AIE state (steps 108-114) as will be described hereinbelow in more detail. More specifically, the space-partitioning grid allows optimizing the search for barriers that is necessary when an Avoid Barriers behaviour is activated and is also used by the surface-hugging and collision subsolvers, which will be described hereinbelow.

**[00124]** Increasing the number of partitions in the grid generally decreases the computational time needed to update the world, but increases

the solver memory usage. The space-partitioning grid is defined via the Grid parameters of the AI solver. The number of partitions along each axis may be specified which effectively divides the world into a given number of cells. Choosing suitable values for these parameters allows tuning the performance. However, values that are too large or too small can have a negative impact on performance. Cell size should be chosen based on average barrier size and density and should be such that, on average, each cell holds about 4 or 5 barriers.

**[00125]** The solver of the digital world model includes subsolvers, which are the various engines of the solver that are used to run the simulation. Each subsolver manages a particular aspect of object and AIE simulation in order to optimize computations.

**[00126]** After the digital world has been set, autonomous image entities (AIE) are defined in step 104. Each AIE may represent a character or an object that is characterized by attributes defining the AIE relatively to the image objects elements of the digital world, and behaviours for modifying some of the attributes. Each AIE is associated to animation clips allowing representing the AIE in movement in the digital world. Virtual sensors allow the AIE to gather data information about image object elements or other AIE within the digital world. Decision trees are used for processing the data information resulting in selecting and triggering one of the animation cycle or selecting a new behaviour.

**[00127]** As it is believed to be well known in the art, an animation cycle, which will also be referred to herein as “animation clip” is a unit of animation that typically can be repeated. For example, in order to get a character to walk, the animator creates a “walk cycle”. This walk cycle makes the character walks one iteration. In order to have the character walk more, more iterations of the cycle are played. If the character speeds up or slows



down during time, the cycle is “scaled” accordingly so that the cycle speed matches the character displacement so that there is no slippage (e.g., it looks like the character is slipping on the ground).

**[00128]** The autonomous image entities are tied to transform nodes of the animating engine (or platform). The nodes can be in the form of locators, cubes or models of animals, vehicles, etc. Since animation clips and transform nodes are believed to be well known in the art, they will not be described herein in more detail.

**[00129]** Figure 4 shows a co-ordinate system for the AIE and used by the solver.

**[00130]** Examples of an AIE attributes are briefly described in the following tables. Even though, this table refers to characters, the listed attributes apply to all AIE.

Attribute	Description
<i>Exists</i>	This attribute allows the solver whether or not to consider the AIE in the world. If this attribute it is set to <i>off</i> , the solver ignores the AIE and does not update it. This attribute allows dynamically creating and killing characters (AIEs).
<i>Hug Terrain</i>	This attribute allows setting whether or not the AIE will hug the terrain. If this is set to <i>on</i> the AIE will remain on the terrain. It is to be noted that terrains are activated only when the solver is in 2D mode.
<i>Align Terrain Normal</i>	This attribute allows setting whether or not the AIE will align with the terrain’s surface normal. This parameter is taken into account when the AIE is hugging the terrain.
<i>Terrain Offset</i>	This attribute specifies an extra height that will be given to a character when it is on a terrain. The offset is only taken into account when the AIE is hugging the terrain. A positive value causes the AIE to float above the terrain, and a negative value causes the AIE to be sunken into the terrain.

Attribute	Description
<i>Hug Surface</i>	This attribute specifies whether or not the AIE will hug a surface. A surface is a barrier with the <i>Surface</i> attribute set to true. Surface hugging applies in a 3D solver. The AIE hugs the nearest surface below it.
<i>Align Surface Normal</i>	This attribute specifies whether or not the AIE's up orientation aligns to the surface normal. This parameter is taken into account when the AIE is on a surface. An AIE with both hug surface and align surface enabled will follow a 3D surface defined by barriers, while aligning the up of the AIE according to the surface.
<i>Surface Offset</i>	This attribute specifies an extra height that will be given to an AIE when it is on a surface. The offset is taken into account only when the AIE is hugging a surface. A positive value will cause the AIE to float above the surface, and a negative value will cause the AIE to be sunken into the surface.
<i>Collidable</i>	This attribute specifies whether or not collisions with other collidable objects will be resolved. If this parameter is set to false then nothing will prevent the AIE from occupying the same space as other objects, as would (for instance) a ghost.
<i>Radius</i>	This attribute specifies the radius of the AIE's bounding sphere. Since the concept of bounding sphere is believed to be well known in the art, it will not be described herein in more detail.
<i>Right Turning Radius</i>	This attribute specifies the maximum right turning angle (clockwise yaw) per frame measured in degrees. The angle can range from 0-180 degrees.
<i>Left Turning Radius</i>	This attribute specifies the maximum left turning angle (anticlockwise yaw) per frame measured in degrees. The angle can range from 0-180 degrees.
<i>Up Turning Radius</i>	This attribute specifies the maximum up turning angle (positive pitch) per frame measured in degrees. The angle can range from 0-180 degrees.
<i>Down Turning Radius</i>	This attribute specifies the maximum down turning angle (negative pitch) per frame measured in degrees. The angle can range from 0-180 degrees.
<i>Maximum Angular Acceleration</i>	This attribute specifies the maximum positive change in angular speed of the AIE, measured in degrees/frame <sup>2</sup> . If this variable is larger than the turning radii, it will have no effect. If set smaller than the turning radii, it will increase the AIE's resistance to angular change. In general, the maximum angular acceleration should be set smaller than the maximum angular deceleration to avoid overshoot and oscillation effects.

Attribute	Description
<i>Maximum Angular Deceleration</i>	This attribute specifies the maximum negative change in angular speed of the character, measured in degrees/frame <sup>2</sup> . If this variable is larger than the turning radii, it will have no effect. If set smaller than the turning radii, it will increase the AIE's resistance to angular change. In general, the maximum angular acceleration should be set smaller than the maximum angular deceleration to avoid overshoot and oscillation effects.
<i>Maximum Pitch (a.k.a. Max Stability Angle)</i>	This attribute specifies the maximum angle of deviation from the z-axis that the object's top vector may have, measured in degrees. The maximum pitch can range from -180 to 180 degrees. This attribute can be used to limit how steep a hill the AIE can climb or descend to prevent objects from incorrectly turning upside down.
<i>Maximum Roll</i>	This attribute specifies the maximum angle of deviation from the x-axis that the object's top vector may have, measured in degrees. The maximum can roll range from -180 to 180 degrees. This attribute can be used to limit the side-to-side tilting of the AIE to prevent objects from incorrectly turning upside down.
<i>Min Speed</i>	This attribute specifies the minimum speed (distance units/frame) of the AIE.
<i>Max Speed</i>	This attribute specifies the maximum speed (distance units/frame) of the AIE.
<i>Max Acceleration</i>	This attribute specifies the maximum positive change in speed (distance units/frame <sup>2</sup> ) of the AIE.
<i>Max Deceleration</i>	This attribute specifies the maximum negative change in speed (distance units/frame <sup>2</sup> ) of the AIE.

Attribute	Description
<i>Brake Padding</i> and <i>Braking Softness</i>	<p>Braking is only applied when an AIE tries to turn at an angle greater than one of its turning radii. When this occurs, the <i>Brake Padding</i> and <i>Braking Softness</i> parameters work together to slow the AIE down so that it doesn't overshoot the turn.</p> <p><i>Brake Padding</i> controls when braking is applied. It can be set to 0, which means that braking will be applied as soon as the object tries to turn beyond one of its maximum turning radii, or 1 which means that braking is never applied. Values between 0 and 1 interpolate those extremes. The default value is 0.</p> <p><i>Braking softness</i> controls the gentleness of braking and can be set to any positive number, including zero. A value of 0 corresponds to maximum braking strength and the AIE will come to a complete stop as soon as the brakes are applied. A value of 1 corresponds to normal strength, and values greater than 1 result in progressively gentler braking. The default value is 1.</p> <p>Setting a very large <i>Braking Softness</i> (effectively <math>+\infty</math>) is equivalent to setting the <i>Brake Padding</i> to 1, which is equivalent to turning braking off.</p>
<i>Forward Motion Only</i>	<p>This attribute is set to <i>on</i> to limit the movement of the AIE such that it may only move in the direction it is facing. <i>Off</i> will allow the AIE to move and face in different directions, provided that its behaviours are set up to produce such motion. The default value is <i>on</i>.</p>
<i>Initial Speed</i>	<p>This attribute specifies the initial speed of the AIE (distance units/frame) at start time.</p>
<i>Initial Position</i> X, Y, Z	<p>This attribute specifies the initial position of the AIE at start time. The default is the position where the object was created.</p>
<i>Initial Orientation</i> X, Y, Z	<p>This attribute specifies the initial orientation of the AIE at start time. The default is the orientation of the object when created.</p>
<i>Display Radius</i>	<p>This attribute specifies whether or not the radius and heading of the AIE will be displayed.</p>
<i>Current Speed</i>	<p>This attribute specifies the current speed (distance units/frame) of the AIE. The solver controls this variable.</p>
<i>Translate</i>	<p>This attribute specifies the current position of the AIE. The AI solver controls this variable.</p>
<i>Rotate</i>	<p>This attribute specifies the current orientation of the AIE. The solver controls this variable.</p>

**[00131]** Of course, other attributes can also be used to characterize an AIE.

**[00132]** In step 106, each AIE attributes are initialized and an initial behaviour among the set of behaviours defined for each AIE is assigned thereto. The initialisation of attributes may concern only selected attributes, such as the initial position of the AIE, its initial speed, etc. As described in the above table, some attributes are modifiable by the solver or by a user via a user interface or a keyable command, for example when the method 100 is embodied in a computer game.

**[00133]** The concept of AIE behaviour will now be described hereinbelow in more detail.

**[00134]** In addition to attributes, AIE from the present invention are also characterized by behaviours. Along with the decision trees, the behaviours are the low-level thinking apparatus of an AIE. They take raw input from the digital world using virtual sensors, process it, and change the AIE's condition accordingly.

**[00135]** Behaviours can be categorized, for example, as State Change behaviours and Locomotive behaviours. State change behaviours modify a character's internal state attributes, which represent for example the AIE's "health", or "aggressivity", or any other non-apparent characteristics of the AIE. Locomotive behaviours allow an AIE to move. These locomotive behaviours generate steering forces that can affect any or all of an AIE's direction of motion, speed, and orientation (i.e. which way the AIE is facing) for example.

**[00136]** The following table includes examples of behaviours:

Simple behaviours:

- Avoid Barriers
- Avoid Obstacles
- Accelerate At
- Maintain Speed At
- Wander Around
- Orient To

Targeted behaviours:

- Seek To
- Flee From
- Look At
- Follow Path
- Seek To Via Network

Group behaviours:

- Align With
- Join With
- Separate From
- Flock With

State Change behaviours:

- State Change On Proximity
- Target State Change On Proximity

**[00137]** A locomotive behaviour can be seen as a force that acts on the AIE. This force is a behavioural force, and is analogous to a physical force (such as gravity), with a difference that the force seems to come from within the AIE itself.

**[00138]** It is to be noted that behavioural forces can be additive. For example, an autonomous character may simultaneously have more than one active behaviours. The solver calculates the resulting motion of the character by combining the component behavioural forces, in accordance with behaviour's priority and intensity. The resultant behavioural force is then applied to the character, which may impose its own limits and constraints (specified by the character's turning radius attributes, etc) on the final motion.

**[00139]** The following table briefly describes examples of parameters that can be used to define behaviours:

Parameter	Description
<i>Active</i>	This parameter defines whether or not the behaviour is active. If the behaviour is not active it will be ignored by the solver and will have no effect on the AIE.

Parameter	Description
<i>Intensity</i>	The higher the intensity, the stronger the behavioural steering force. The lower the intensity (or the closer to 0), the weaker the behavioural steering force. For example, an intensity of 1 causes the behaviour to operate at "full strength", an intensity of 2 causes the behaviour to produce twice the steering force, and an intensity of 0 effectively turns the behaviour off.
<i>Priority</i>	<p>The priority of a behaviour defines the precedence it will take over other behaviours. Behaviours of higher priority (<i>i.e.</i> those with a lower numerical value) take precedence over behaviours of lower priority. Therefore, if a behaviour of higher priority produces a desired motion, then a behaviour of lower priority is ignored. A priority of 0 is considered the highest priority (<i>i.e.</i> of most importance).</p> <p>For example, a character has a Flee From behaviour with priority 0 and a Follow Path behaviour with priority 1. If the Flee From behaviour produces a desired motion, then the Follow Path behaviour is ignored. However, if the Flee From Behaviour does not produce a desired motion, such as when it is inactive or the target is outside the activation radius, the Follow Path behaviour is taken into account.</p>
<i>Blend Time</i>	This parameter allows controlling the transition time, expressed as number of frames, that a behaviour will take to change from an active to inactive state or vice-versa. For example, a blend time of zero means that a behaviour can change its state instantaneously. In other words the behaviour could be inactive one frame and at full-force the next. Increasing the blend time will allow behaviours to fade in and out, thus creating smoother transitions between behaviour effects. However, this will also increase the time required for an AIE to respond to stimuli provided by one of the AIE's sensor as will be described hereinbelow in more detail.
<i>Affects Speed</i>	This parameter indicates whether the behavioural force produced by this behaviour may affect the speed of a moving AIE. This parameter is set to <i>on</i> by default. If a speed force is produced by the behaviour, behaviours of a lower priority are prevented from affecting the speed of the AIE.
<i>Affects Direction</i>	This parameter indicates whether the behavioural force produced by this behaviour may affect the direction of motion of an AIE. By default this parameter is set to <i>on</i> . If a directional force is produced by the behaviour, behaviours of a lower priority are prevented from affecting the direction of the AIE.

Parameter	Description
<i>Affects Orientation</i>	This parameter indicates whether the behavioural force produced by this behaviour may affect the orientation of an AIE (which way the AIE is facing, as opposed to its direction of motion). By default this parameter is set to <i>on</i> . If an orientation force is produced by the behaviour, behaviours of a lower priority are prevented from affecting the orientation of the AIE.

**[00140]** The behaviours allow creating a wide variety of actions for AIEs. Behaviours can be divided into four subgroups: simple behaviours, targeted behaviours, group behaviours and state change behaviours.

**[00141]** Simple behaviours are behaviours that only involve a single AIE.

**[00142]** Targeted behaviours apply to an AIE and a target object, which can be any other object in the digital world (including groups of objects).

**[00143]** Group behaviours allow AIEs to act and move as a group where the individual AIEs included in the group will maintain approximately the same speed and orientation as each other.

**[00144]** State change behaviours enable the state of an object to be changed.

**[00145]** Examples of behaviours will now be provided in each of the four categories. Of course, it is believed to be within a person skilled in the art to provide an AIE with other behaviours.

**[00146]** Simple Behaviours

**[00147]** Avoid Barriers



**[00148]** The Avoid Barriers behaviour allows a character to avoid colliding with barriers. When barriers are defined in the world, the space-partitioning grid in the AI solver may be specified in order to optimize the solver calculations that concern barriers.

**[00149]** Parameters specific to this behaviour may include, for example:

Parameter	Description
<i>Avoid Distance</i>	The distance from a barrier at which the AIE will attempt to avoid it. This is effectively the distance at which barriers are visible to the AIE.
<i>Avoid Distance Is Speed Adjusted</i>	Whether or not the avoidance distance is adjusted according to the AIE's speed. If this is set to <i>on</i> , the faster the AIE moves, the greater the avoidance distance.
<i>Avoid Width Factor</i>	The avoidance width factor defines how wide the "avoidance capsule" is (the length of the "avoidance capsule" is equal to the Avoid Distance). If a barrier lies within the avoidance capsule, the AIE will take evasive action. The value of the avoidance width factor is multiplied by the AIE's width in order to determine the true width (and height in a 3D solver) of the capsule. A value of 1 sets the capsule to the same width as the AIE's diameter.
<i>Barrier Repulsion Force</i>	Allows controlling how much the AIE is pushed away from barriers. A value of 0 indicates no repulsion and the AIE will tend to move parallel to nearby barriers. Larger values will add a component of repulsion based on the AIE's incident angle.
<i>Avoidance Queuing</i>	Allows controlling the AIE's barrier avoidance strategy. If set to <i>on</i> the AIE will slow down when approaching a barrier, if set to <i>off</i> the AIE will dodge the barrier. The default value is <i>off</i> .

**[00150]** The Avoid Obstacles behaviour allows an AIE to avoid colliding with obstacles, which can be other autonomous and non-autonomous image entities. Similar parameters than those detailed for the Avoid Barriers behaviour can also be used to define this behaviour.

**[00151]            Accelerate At**

**[00152]**            The Accelerate At behaviour attempts to accelerate the AIE by the specified amount. For example, if the amount is a negative value, the AIE will decelerate by the specified amount. The actual acceleration/deceleration may be limited by max acceleration and max deceleration attributes of the AIE.

**[00153]**            A parameter specific to this behaviour is the Acceleration, which represents the change in speed (distance units/frame<sup>2</sup>) that the AIE will attempt to maintain.

**[00154]            Maintain Speed At**

**[00155]**            The Maintain Speed At behaviour attempts to set the target AIE's speed to a specified value. This can be used to keep a character at rest or moving at a constant speed. If the desired speed is greater than the character's maximum speed attribute, then this behaviour will only attempt to maintain the character's speed equal to its maximum speed. Similarly, if the desired speed is less than the character's minimum speed attribute, this behaviour will attempt to maintain the character's speed equal to its minimum speed.

**[00156]**            A parameter allowing defining this behaviour is the desired speed (distance units/frame) that the character will attempt to maintain.

**[00157]            Wander Around**

**[00158]**            The Wander Around behaviour applies random steering forces to the AIE to ensure that it moves in a random fashion within the solver area.

**[00159]** Parameters allowing to define this behaviour may be for example:

Parameter	Description
<i>Is Persistent</i>	This parameter allows defining whether or not the desired motion calculated by this behaviour is applied continuously (at every frame) or only when the desired motion changes (see the <i>Probability</i> attribute). A persistent Wander Around behaviour produces the effect of following random waypoints. A non-persistent Wander Around behaviour causes the AIE to slightly change its direction and/or speed when the desired motion changes.
<i>Probability</i>	This parameter allows defining the probability that the direction and/or speed of wandering will change at any time frame. For example, a value of 1 means that it will change each time frame, a value of 0 means that it will never change. On average, the desired motion produced by this behaviour will change once every $1/\text{probability}$ frames ( <i>i.e.</i> average frequency = $1/\text{probability}$ )
<i>Max Left Turn</i>	This parameter allows defining the maximum left wandering turn angle in degrees at any time frame.
<i>Max Right Turn</i>	This parameter allows defining the maximum right wandering turn angle in degrees at any time frame.
<i>Left Right Turn Radii Noise Frequency</i>	This parameter affects the value of the pseudo-random left and right turn radii generated by this behaviour. A valid range can be between 0 and 1. The higher the frequency the more frequent an AIE will change direction. The lower the frequency the less often an AIE will change direction.
<i>Max Up Turn</i>	This parameter allows defining the maximum up wandering turn angle in degrees at any time frame.
<i>Max Down Turn</i>	This parameter allows defining the maximum down wandering turn angle in degrees at any time frame.
<i>Up Down Turn Radii Noise Frequency</i>	This parameter affects the value of the pseudo-random up and down turn radii generated by this behaviour. The valid range is between 0 and 1. The higher the frequency the more frequent an AIE will change direction. The lower the frequency the less often an AIE will change direction.
<i>Max Deceleration</i>	This parameter allows defining the maximum wander deceleration (distance units/frame <sup>2</sup> ) at any time frame.
<i>Max Acceleration</i>	This parameter allows defining the maximum wander acceleration (distance units/frame <sup>2</sup> ) at any time frame.

Parameter	Description
<i>Speed Noise Frequency</i>	This parameter affects the value of the pseudo-random speed generated by this behaviour. The valid range is between 0 and 1. The higher the frequency the more frequent an AIE will change direction. The lower the frequency the less often an AIE will change direction.
<i>Min Speed</i>	This parameter allows defining the minimum speed (distance units/frame) that the behaviour will attempt to maintain.
<i>Use Min Speed</i>	This parameter allows defining whether or not the <i>Min Speed</i> attribute will be used.
<i>Max Speed</i>	This parameter allows defining the maximum speed (distance units/frame) that the behaviour will attempt to maintain.
<i>Use Max Speed</i>	This parameter allows defining whether or not the <i>Max Speed</i> attribute will be used.

**[00160]**            Orient To

**[00161]**            The Orient To behaviour allows an AIE to attempt to face a specific direction.

**[00162]**            Parameters allowing to define this behaviour are:

Parameter	Description
<i>Desired Forward Orientation</i>	This parameter allows defining the direction this AIE will attempt to face. For example, a desired forward orientation of (1,0,0) will make an AIE attempt to align itself with the x-axis. When a 2D solver is used, the y component of the desired forward orientation is ignored.
<i>Relative</i>	If true, then the desired forward orientation attribute is interpreted to be relative to the current character forward. If false, then the desired forward is in absolute world coordinates. By default, this value is set to false.

**[00163]**            Targeted Behaviours

**[00164]** The following behaviours apply to an AIE (the source) and another object in the world (the target). Target objects can be any object in the world such as autonomous or non-autonomous image entities, paths, groups and data. If the target is a group, then the behaviour applies only to the nearest member of the group at any one time. If the target is a datum, then it is assumed that this datum is of type ID and points to the true target of the behaviour. An ID is a value used to uniquely identify objects in the world. The concept of datum will be described in more detail hereinbelow.

**[00165]** The following parameters, shared by all targeted behaviours, are:

Parameter	Description
<i>Activation Radius</i>	The <i>Activation Radius</i> determines at what point the behaviour is triggered. The behaviour will only be activated and the AIE will only actively seek a target if the AIE is within the activation radius distance from the target. A negative value for the activation radius indicates that there is no activation radius, or that the feature is not being used. This means that the behaviour will always be <i>on</i> regardless of the distance between the AIE and the target.
<i>Use Activation Radius</i>	This parameter allows defining whether or not the <i>Activation Radius</i> feature will be used. If this is <i>off</i> , the behaviour will always be activated regardless of the location of the AIE.

**[00166]** Seek To

**[00167]** The Seek To behaviour allows an AIE to move towards another AIE or towards a group of AIEs. If an AIE seeks a group, it will seek the nearest member of the group at any time.

**[00168]** Parameters allowing to define this behaviour are for example:

Attribute	Description
<i>Look Ahead Time</i>	This parameter instructs the AIE to move towards a projected future point of the object being sought. Increasing the amount of look-ahead time does not necessarily make the Seek To behaviour any "smarter" since it simply makes a linear interpolation based on the target's current speed and position. Using this parameter gives the behaviour sometimes referred to as "Pursuit".
<i>Offset radius</i>	This parameter allows specifying an offset from the target's centre point that the AIE will actually seek towards.
<i>Offset Yaw Angle</i>	<p>This parameter allows defining the angle in degrees about the front of the target in the yaw direction that the offset is calculated. The angle describes the amount of counter-clockwise rotation about the front of the target. For example, to make a soldier follow a leader, the soldier seek the leader with a positive offset radius and an offset yaw angle of 180°.</p> <p>This attribute is ignored if the <i>Strafing</i> parameter is turned on. Strafing automatically sets an appropriate value for the offset angle.</p>
<i>Offset Pitch Angle</i>	This parameter is the similar to <i>Offset Yaw Angle</i> but for the offset angle in the pitch direction relative to the target object's orientation. This applies only in the case of a 3D solver and will be ignored in a 2D solver.
<i>Contact Radius</i>	This parameter allows specifying a proximity radius at which point the behaviour is triggered. In other words, it defines the point at which the AIE has reached the target and has no reason to continue seeking it. If the parameter is set to -1, this feature is turned <i>off</i> and the AIE will always attempt to seek the target regardless of their relative positions. Since the contact radius extends the target's radius, a value of 0 means that the AIE will stop seeking when it touches (or intersects with) the target.
<i>Use Contact Radius</i>	This parameter allows defining whether or not the <i>Contact Radius</i> feature is used. If this is <i>off</i> , the AIE will always attempt to seek the target regardless of their relative positions
<i>Slowing Radius</i>	The slowing radius specifies the point at which the AIE begins to attempt to slow down and arrive at a standstill at the contact radius (or earlier). If set to -1, this feature is turned <i>off</i> and the AIE will never attempt to stop moving when it reaches its target. This feature of Seek To is sometimes referred to as "Arrival". It is to be noted that the slowing radius is taken to be the distance from the contact radius, which itself is the distance from the external radius of the target.
<i>Use Slowing Radius</i>	This parameter allows defining whether or not the <i>Slowing Radius</i> feature is used. If this is <i>off</i> , the AIE will not attempt to slow down when reaching the target.

Attribute	Description
<i>Desired Speed</i>	The desired speed instructs an AIE to move towards the target at the specified speed. If this is set to a negative number or <i>Use Desired Speed</i> is <i>off</i> , this feature is turned off and the AIE will attempt to approach the target at its maximum speed.
<i>Use Desired Speed</i>	This parameter allows defining whether or not the <i>Desired Speed</i> attribute will be used. If this is <i>off</i> , the AIE will attempt to approach the target at its maximum speed.

#### [00169] Flee From

[00170] The Flee From behaviour allows an AIE to flee from another AIE or from a group of AIEs. When an AIE flees from a group, it will flee from the nearest member of the group at any time. The Flee From behaviour has the same attributes as the Seek To behaviour, however, it produces the opposite steering force. Since the parameters allowing defining the Flee From behaviour are very similar to those of the Seek To behaviour, they will not be described herein in more detail.

#### [00171] Look At

[00172] The Look At behaviour allows an AIE to face another AIE or a group of AIEs. If the target of the behaviour is a group, the AIE attempts to look at the nearest member of the group.

#### [00173] Strafe

[00174] The Strafe behaviour causes the AIE to “orbit” its target, in other words to move in a direction perpendicular to its line of sight to the target. A probability parameter allows to determine how likely it is at each frame that the AIE will turn around and start orbiting in the other direction. This can be used, for instance, to make a moth orbit a flame.

**[00175]** For example, the effect of a guard walking sideways while looking or shooting at its target can be achieved by turning off the guard's Forward Motion Only property, and adding a Look At behaviour set towards the guard's target. It is to be noted that, to do this, Strafe is set to Affects direction only, whereas Look At is set to Affects orientation only.

**[00176]** A parameter specific to this behaviour may be, for example, the Probability, which may take a value between 0 and 1 that determines how often the AIE change direction of orbit. For example, at 24 frames per second, a value of 0.04 will trigger a random direction change on average every second, whereas a value of 0.01 will trigger a change on average every four seconds.

**[00177]** Go Between

**[00178]** The Go Between behaviour allows an AIE to get in-between the first target and a second target. For example this behaviour can be used to enable a bodyguard character to protect a character from a group of enemies.

**[00179]** The following parameter allow specifying this behaviour:, which may take a value between 0 and 1 that determines how close to the second target you want to go.

**[00180]** Follow Path

**[00181]** The Follow Path behaviour allows an AIE to follow a path. For example this behaviour can be used to enable a racecar to move around a racetrack.

**[00182]** The following parameters allow defining this behaviour:



Parameter	Description
<i>Use Speed Limits</i>	This parameter allows defining whether or not the AIE will attempt to use the speed limits of the waypoints on the path. If this parameter is set to <i>off</i> , the AIE will attempt to follow the path at its maximum speed.
<i>Path Is Looped</i>	This parameter allows defining whether or not the AIE will go to the first waypoint when it reaches the last waypoint. If the parameter is set to <i>off</i> , when the AIE reaches the last waypoint it will hover around that waypoint.

### **[00183]**            Seek To Via Network

**[00184]**            The Seek To Via Network behaviour can be viewed as an extension of the Seek To behaviour that allows a source (AIE) to use a waypoint network to navigate towards a target. The purpose of a waypoint network is to store as much pre-calculated information as possible about the world that surrounds the character and, in particular, the position of static obstacles. The waypoint network, which will be described hereinbelow in more detail, can be used for example in one of two ways:

**[00185]**            Edges in the network are used to define a set of “safe corridors” within which a source object can safely navigate without fear of running into a barrier or other static obstacles. Thus, once an AIE has reached a corridor in the network, it can safely navigate from waypoint to waypoint via the network.

**[00186]**            While navigating, periodic reach ability tests are performed in order to determine whether it is safe to cut corners thus producing more natural motion. The frequency of these tests can be adjusted using the behaviour parameters.

**[00187]** In addition to the parameters that are available for the Seek To behaviour, the Seek To Via Network behaviour has the following additional parameters that can be used to control the type and frequency of the vision tests used:

Paramter	Description
<i>Period For Current Location Check</i>	This parameter allows determining how often the AIE's current location is checked. This is to catch situations where an AIE is suddenly transported to another section of the world, e.g. via a teleport or by falling off a cliff. Default value=1. To disable this check, set value to 0.
<i>Period For Target Location Check</i>	This parameter allows determining how often the target's location is checked. This is to handle the case of a dynamic (moving) target. Default value=1. To disable this check, set value to 0.
<i>Period For Path Smoothing Check</i>	This parameter allows determining how often the desired path of the AIE is adjusted to provide "smoother" motion. This is equivalent to looking ahead and checking for shortcuts between the AIE's current location and a future waypoint on the character's desired path. This check is omitted when the vision test to use is set to "Simple". The default value is set 1. To disable this check, set value to 0.
<i>Barrier Padding Factor</i>	The barrier-padding factor is multiplied by an AIE's radius to determine the minimum clearance distance to be used when deciding if an AIE can move around a barrier safely. This value is not used when the vision test to use is set to "Simple". The default value=1.0.

**[00188]** It is to be noted that the Seek To parameters are used to guide the motion of the AIE, however the contact radius and slowing radius parameters are only used when the AIE seeks its final target. In addition, when the AIE seeks its final target, only checks for barrier avoidance are performed rather than checks for current location, target location, and path smoothing. This single check is performed at each call to this behaviour.

**[00189]** Group Behaviours

**[00190]** Group behaviours allow grouping individual AIEs so that they act as a group while still maintaining individuality. Examples include a school of fish, a flock of birds, etc.

**[00191]** The following parameters may be used to define group behaviours:

Parameter	Description
<i>Neighbourhood Radius</i>	This parameter is similar to the "activation radius" in targeted behaviours. The AIE will "see" only those members that are within its neighbourhood radius. The neighbourhood radius is independent of the AIE's radius.
<i>Use Max Neighbours</i>	This parameter allows defining whether or not the <i>Max Neighbours</i> attribute will be used. If this parameter is set to <i>off</i> , then all the group members in the neighbourhood radius are used to calculate the effect of the behaviour.
<i>Max Neighbours</i>	This parameter allows defining the maximum number of neighbours to be used in calculating the effect of the behaviour.

**[00192]** The following includes brief descriptions of examples of group behaviours.

**[00193]** Align With

**[00194]** The Align With behaviour allows an AIE to maintain the same orientation and speed as other members of a group. The AIE may or may not be a member of the group.

**[00195]** Join With

**[00196]** The Join With behaviour allows an AIE to stay close to members of a group. The AIE may or may not be a member of the group.

**[00197]** An example of parameter that can be used to define this behaviour is the Join Distance, which is similar to the "contact radius" in targeted behaviours. Each member of the group within the neighbourhood radius and outside the join distance is taken into account when calculating the steering force of the behaviour. The join distance is the external distance between the characters (i.e. the distance between the outsides of the bounding spheres of the characters). The value of this parameter determines the closeness that members of the group attempt to maintain.

**[00198]** Separate From

**[00199]** The Separate From behaviour allows an AIE to keep a certain distance away from members of a group. For example, this can be used to prevent a school of fish from becoming too crowded. The AIE to which the behaviour is applied may or may not be a member of the group.

**[00200]** The Separation Distance is an example of parameters that can be used to define this behaviour. Each member of the group within the neighbourhood radius and inside the separation distance will be taken into account when calculating the steering force of the behaviour. The separation distance is the external distance between the AIEs (i.e. the distance between the outsides of the bounding spheres of the AIEs). The value of this parameter determines the external separation distance that members of the group will attempt to maintain.

**[00201]** Flock With

**[00202]** This behaviour allows AIEs to flock with each other. It combines the effects of the Align With, Join With, and Separate From behaviours.

**[00203]** The following table describes parameters that can be used to define this behaviour:

Parameter	Description
<i>Alignment Intensity</i>	This parameter allows defining the relative intensity of the Align With behaviour.
<i>Join Intensity</i>	This parameter allows defining the relative intensity of the Join With behaviour.
<i>Separation Intensity</i>	This parameter allows defining the relative intensity of the Separate From behaviour.
<i>Join Distance</i>	This parameter determines the closeness that members of the group will attempt to maintain.
<i>Separation Distance</i>	This parameter determines the external separation distance that members of the group will attempt to maintain.

**[00204]** State Change Behaviours

**[00205]** State Change behaviours allow changing AIEs' states. Examples of State Change behaviours will now be provided.

**[00206]** State Change On Proximity

**[00207]** The State Change On Proximity behaviour allows an AIE's state to be changed based on its distance from a target. For example, the "alive" state of a soldier can be change to false once an enemy kills him.

**[00208]** Examples of parameters allowing defining the State Change On Proximity behaviour:

Parameter	Description
<i>Trigger Radius</i>	This parameter allows defining the external distance between the two AIEs at which the State Change behaviour is triggered.

Parameter	Description
<i>Probability</i>	This parameter allows defining the probability that the state change is triggered at each frame if the AIEs are within the trigger radius. The value ranges between 0 and 1. 0 means that the state change will not occur and 1 means that the state change will definitely occur.
<i>Changing State</i>	This parameter allows defining the state of the source character to be changed.
<i>Change Action</i>	This parameter is assigned one of the following values: <ul style="list-style-type: none"> <li>- <i>AbsoluteValue</i>: sets the state to the <i>Change Value</i>.</li> <li>- <i>AbsoluteBoolean</i>: assumes the <i>Change Value</i> is a Boolean and changes the state to that.</li> <li>- <i>ToggleBoolean</i>: assumes the state is a Boolean value and toggles it.</li> <li>- <i>Increment</i>: Increments the value of the state by 1.</li> <li>- <i>Decrement</i>: Decrements the value of the state by 1.</li> </ul>
<i>Change Value</i>	This parameter allows defining the new value of the state.
<i>Use Default Value</i>	This parameter allows defining whether or not the value of the state will be set to the default value if the target does not exist or if the target is outside the activation radius.
<i>Default Value</i>	If <i>Use Default Value</i> is <i>on</i> , then the value of the state will be set to this value if the target does not exist or if the target is outside the activation radius.

#### [00209] Target State Change On Proximity

[00210] The Target State Change On Proximity behaviour is similar to the State Change On Proximity behaviour with a difference that it affects the target character's state. For example, a shark kills a fish (i.e. change the fish's "alive" state to false) as soon as the shark is within a few centimetres of the fish.

[00211] The following table includes examples of parameters that can be used to define this behaviour:

Parameter	Description
<i>Trigger Radius</i>	This parameter allows defining the external distance between the two AIEs at which the state change behaviour is triggered.
<i>Probability</i>	This parameter allows defining the probability of the state change being triggered at each frame if the AIEs are within the trigger radius. The value ranges between 0 and 1. 0 means that the state change will not occur and 1 means that the state change will definitely occur.
<i>Changing State</i>	This parameter allows defining the state of the target AIE to be changed.
<i>Change Action</i>	This parameter can take any of the following values: <ul style="list-style-type: none"> <li>- <i>AbsoluteValue</i>: sets the state to the <i>Change Value</i>.</li> <li>- <i>AbsoluteBoolean</i>: assumes the <i>Change Value</i> is a Boolean and changes the state to that.</li> <li>- <i>ToggleBoolean</i>: assumes the state is a Boolean value and toggles it.</li> <li>- <i>Increment</i>: Increments the value of the state by 1.</li> <li>- <i>Decrement</i>: Decrements the value of the state by 1.</li> </ul>
<i>Change Value</i>	This parameter allows defining the new value of the state.
<i>Use Default Value</i>	This parameter allows defining whether or not the value of the state will be set to the default value if the target does not exist or if the target is outside the activation radius.
<i>Default Value</i>	<i>Use Default Value</i> is <i>on</i> , then the value of the state will be set to this value if the target does not exist or if the target is outside the activation radius.

## [00212] Combining Behaviours

**[00213]** An AIE can have multiple active behaviours associated thereto at any given time. Since the possibility that these behaviours be in conflict with each other could arise, the method and system for on-screen animation of digital entities according to the present invention provides means to assign importance to a given behaviour.

**[00214]** A first means to achieve this is by assigning intensity and priority to a behaviour. The assigned intensity of a behaviour affects how strong the steering force generated by the behaviour will be. The higher the intensity the greater the generated behavioural steering forces. The priority of a behaviour defines the precedence the behaviour should have over other behaviours. When a behaviour of a higher priority is activated, those of lower priority are effectively ignored. By assigning intensities and priorities to behaviours the animator informs the solver which behaviours are more important in which situations in order to produce a more realistic animation.

**[00215]** In order for the solver to calculate the new speed, position, and orientation of an AIE, the solver calculates the desired motion of all behaviours, sums up these motions based on each behaviour's intensity, while ignoring those with lower priority, and enforces the maximum speed, acceleration, deceleration, and turning radii defined in the AIE's attributes. Finally, braking due to turning may be taken into account. Indeed, based on the values of the character's Braking Softness and Brake Padding attributes, the character may slow down in order to turn.

**[00216]** Providing, for example, the case of a school of fish and a hungry shark in a large aquarium, and more specifically the case where a fish wants to escape the hungry shark. At this point in time both the fish's "Flee From" shark and "Flock With" other fish behaviours will be activated causing two steering forces to act on the fish in unison. Therefore, the fish tries to escape the shark and stay with the other fish at the same time. The resulting active steering force on the fish will be the weighted sum of the individual behavioural forces, based on their intensities. For example, for the fish, it is much more important to flee from the shark than to stay in a school formation. Therefore, a higher intensity is assigned to the fish's "Flee From" behaviour than to the "Flock With" behaviour. This allows the fish to break formation when



trying to escape the shark and then to regroup when it is far enough away from the shark.

**[00217]** Although the resulting behaviour can be achieved simply by adjusting intensities, ideally when the fish sees the shark it would disable its “Flock With” behaviour and enable its “Flee From” behaviour. Once out of range of the shark, the fish would then continue to swim in a school by disabling its “Flee From” behaviour and enabling its “Flock With” behaviour. This type of behavioural control can be achieved by setting the behaviours’ priorities. By giving the “Flee From” behaviour a higher priority than the “Flock With” behaviour, when a fish is fleeing from a shark, its “Flock With” behaviour will be effectively disabled. Therefore, a fish will not try to remain with the other fish while trying to flee the shark, but once it has escaped the shark its “Flock With” behaviour will be reactivated and the fish will regroup with its school.

**[00218]** In many relatively simple cases such as described in this last example, to obtain a realistic animation sequence it is usually sufficient to assign various degrees of intensities and priorities to specific behaviours. However, in a more complicated scenario, simply tweaking a behaviour’s attributes may not produce acceptable results. In order to implement higher-level behaviour, an AIE needs to be able to make decisions about what actions to take according to its surrounding environment. The following section describes how an AIE uses sensors to gather data information image object elements or other AIE in the digital world (step 108) and how decisions are made and action selected based on this information (step 110).

**[00219]** For example, to cause a character to move along a path but run away from any nearby enemies, the following logic can be implemented:

**[00220]** if an enemy is near,

**then:** run away

**else:** follow the path

**[00221]** This relatively simple piece of logic can be divided as follows:

**[00222]** 1. The conditional: "if enemy is near".

**[00223]** 2. The Actions: "run away", or "follow the path", depending of the current state of the conditional.

**[00224]** In the method 100, the conditional is implemented by creating a Sensor, which will output its findings to an element of the character's memory called a Datum.

**[00225]** The Actions are implemented using Commands. Commands can be used to activate behaviours or animation cycles, to set character attributes, or to set Datum values. In this example, the commands would activate a FleeFrom behaviour or a FollowPath behaviour for example.

**[00226]** Finally, a Decision Tree is used to group the Actions with the Conditional. A Decision Tree allows nesting multiple conditional nodes in order to produce logic of arbitrary complexity.

**[00227]** Data Information

**[00228]** An AIE's data information can be thought of as its internal memory. Each datum is an element of information stored in the AIE's internal memory. For example, a datum could hold information such as whether or not an enemy is seen or who is the weakest ally. A Datum can also be used as a state variable for an AIE.

**[00229]** Data are written to by a character's Sensors, or by Commands within a Decision Tree. The Datum's value is used by the Decision Tree to activate and deactivate behaviours and animations, or to test the character's state. Sensors and Decision trees will be described hereinbelow in more detail.

**[00230]**        Sensors

**[00231]**        AIEs use sensors to gain information about the world. A sensor will store its sensed information in a datum belonging to the AIE.

**[00232]**        A parameter can be used to trigger the activation of a sensor. If a sensor is set off, it will be ignored by the solver and will not store information in any datum.

**[00233]**        Example of sensors that can be implemented in the method 100 will now be described in more detail. Of course, it is believed to be within the reach of a person skilled in the art to provide additional or alternate sensors depending on the application.

**[00234]**        Vision Sensor

**[00235]**        The vision sensor is the eyes and ears of a character and allows the character to sense other physical objects or AIEs in the virtual world, which can be autonomous or non-autonomous characters, barriers, and waypoints, for example.

**[00236]**        The following parameters allow, for example, defining the vision sensor:

Parameter	Description
<i>Visibility Distance</i>	This parameter allows defining the maximum distance from the AIE that it can sense other objects <i>i.e.</i> how far can the AIE see. The visibility distance is the external distance between the AIE, <i>i.e.</i> the distance between the outsides of the bounding spheres of the AIEs.
<i>Visibility Angles</i>	This parameter allows defining the following four angles: <i>Visibility Right Angle</i> , <i>Visibility Left Angle</i> , <i>Visibility Up Angle</i> , and <i>Visibility Down Angle</i> , specify the field of view of the visibility sensor measured in degrees. Any object outside the frustum defined by these angles will be ignored.
<i>Can See Through Opaque Barriers</i>	If this parameter is set to <i>off</i> , then the sensor will not sense objects behind opaque barriers.
<i>Object Type Filter</i>	This parameter allows defining the type of objects this sensor will look for. The options are: All Objects, Barriers, Way Points, or AIEs. For example, if Barriers is chosen then the sensor will only find barriers.
<i>Object Filter</i>	This parameter allows defining the objects this sensor will look for. If this is set to a group, then the sensor will only look for objects in the selected group. If this is set to a path, then the sensor will only look for waypoints on the path. If this is set to a specific object (e.g. a character, a waypoint, or a barrier), then the sensor will ignore all other objects in the world.
<i>Evaluation Function</i>	<p>This parameter allows defining the evaluation function assigns a value to each sensed object. The value of the object, in conjunction with the <i>Min Max</i> attribute, is used to determine the “best” object of all the ones sensed. The possible values are:</p> <ul style="list-style-type: none"> <li>– Any: this chooses the first object sensed. This is the most efficient value of the Evaluation Function. This value could possibly choose the same object every time. If you want a randomly selected object, set the value of the Evaluation Function to “Random”.</li> <li>– Distance: this chooses an object based on its distance from the character. If the <i>Min Max</i> attribute is set to minimum, the nearest object to the AIE is chosen. If the <i>Min Max</i> attribute is set to maximum, the furthest object (within the visibility distance) to the AIE is chosen.</li> <li>– Random: this randomly chooses an object.</li> </ul>
<i>Min Max</i>	This parameter allows defining whether the object with the minimum or maximum value is considered the “best” object.

Parameter	Description
<i>Is Any Object Seen Datum</i>	This parameter allows defining the datum that will be used to store whether or not any object was seen, <i>i.e.</i> did the AIE see what it was looking for.
<i>Best Object Datum</i>	This parameter allows defining the datum that will be used to store which “best” object that was sensed, <i>i.e.</i> what exactly did the AIE see.

### **[00237]**            Property Sensor

**[00238]**            The Property sensor is a general-purpose sensor allowing to return and filter the value of any of an AIE’s state, speed, angular velocity, orientation, distances from target, group membership, datum values, bearing, or pitch bearing.

**[00239]**            Unlike other sensors, the property sensor can sense the properties of any AIE in the simulation.

**[00240]**            The following table includes a list of parameters that can be used to define the Property sensor:

Parameter	Description
Property Type	<p>This parameter allows defining the property to be sensed. Options are:</p> <p><b>State:</b> Returns the value of the specified state variable of the targeted AIE.</p> <p><b>Random:</b> Returns a value between the 0 and 1. There is no target AIE for this property type.</p> <p><b>Speed:</b> Returns the current speed of the targeted AIE.</p> <p><b>Angular Velocity:</b> Returns the angular velocity of the targeted AIE. This angle is measured in degrees.</p> <p><b>Distance:</b> Returns the distance from the targeted object.</p> <p><b>Group Membership:</b> Returns whether or not the AIE is a member of the specified group.</p> <p><b>Datum Value:</b> Returns the value of the specified datum.</p> <p><b>Bearing:</b> Returns the difference about the Y axis between the forward orientation of an AIE and the direction of motion of the AIE. The value returned is in degrees.</p> <p><b>Pitch Bearing:</b> Returns the difference about the X axis between the forward orientation of an AIE and the direction of motion of the AIE. The value returned is in degrees.</p> <p><b>Surface Slope:</b> Returns the angle in degrees between the horizontal and the surface in the direction the AIE is climbing a vertical cliff, -45 for going down a 45° slope).</p>
Target	The following parameter allows defining the AIE whose property is to be sensed. “ <b>Self</b> ” (the default setting) will cause the selected AIE (i.e. the owner of the sensor) to be sensed.
Result Datum	The value sensed is stored in the result datum. For example, a speed sensor will return the speed of an AIE as a float value to the result datum.
Filter Type	Filters are used to evaluate the data returned from a sensor and pass a Boolean value to the Filtered Result Datum.
Is Same As	The Filtered Result Datum will be used to store whether or not the value is exactly the <i>same</i> as the specified value.
Is At Least	The Filtered Result Datum will be used to store whether or not the value is <i>at least</i> the specified value.
Is At Most	The Filtered Result Datum will be used to store whether or not the value is <i>at most</i> the specified value.
Is In Range	The Filtered Result Datum will be used to store whether or not the value is between <i>Minimum</i> and <i>Maximum</i> .

Parameter	Description
Filtered Result Datum	The Boolean result of the filter operation is stored here.

### [00241] Random Sensor

[00242] A random sensor returns a random number within a specified range. The following table includes examples of parameters that allow to define the Random sensor:

Parameter	Description
<i>Minimum</i>	This parameter allows defining the start of the range.
<i>Maximum</i>	This parameter allows defining the end of the range.
<i>Value Datum</i>	This parameter allows defining the datum that will be used to store the random value. If the type attribute of this datum is <i>Boolean</i> , then a random number between 0 and 1 will be generated, and the datum will be set to <i>true</i> if that number falls within the range indicated by the <i>Minimum</i> and <i>Maximum</i> attributes.

### [00243] Value Sensors

[00244] A value sensor allows setting the value of a datum based on whether or not a certain value is within a certain range.

[00245] The following table includes examples of parameters that can be used to define the Value sensor:

Parameter	Description
<i>Minimum</i>	This parameter allows defining the start of the range.
<i>Use Minimum</i>	If this parameter is set to <i>off</i> , the start of the range is considered to be negative infinity,

Parameter	Description
<i>Maximum</i>	This parameter allows defining the end of the range.
<i>Use Maximum</i>	If this parameter is set to <i>off</i> , the end of the range is considered to be infinity.
<i>Is Value In Range Datum</i>	This parameter allows defining the datum that will be used to store whether or not the value is between <i>Minimum</i> and <i>Maximum</i> .

### [00246] Speed Sensor

[00247] A speed sensor is a value sensor that sets the value of a boolean datum based on the speed of the AIE. For example, this sensor can be used to change the animation of an AIE from a walk cycle to a run cycle.

[00248] The Property sensor can be used to read the actual speed of an AIE into a datum.

[00249] The following table includes examples of parameters that can be used to define the Speed sensor:

Parameter	Description
<i>Minimum</i>	This parameter allows defining the start of the range.
<i>Use Minimum</i>	If this parameter is set to <i>off</i> , then the start of the range is considered to be negative infinity,
<i>Maximum</i>	This parameter allows defining the end of the range.
<i>Use Maximum</i>	If this parameter is set to <i>off</i> , then the end of the range is considered to be infinity.
<i>Is Value In Range Datum</i>	This parameter allows defining the datum that will be used to store whether or not the value is between <i>Minimum</i> and <i>Maximum</i> .

### [00250] State Sensor



**[00251]** A state sensor allows setting the value of a boolean datum based on the value of one of the AIE's states. For example, in a battle scene such a sensor can be used to allow AIEs with low health to run away by activating a Flee From behaviour when their "alive" state reaches a low enough value.

**[00252]** The following tables includes examples of parameters that can be used to define a state sensor:

Parameter	Description
<i>State</i>	The following parameter allows defining the state to be used.
<i>Minimum</i>	The following parameter allows defining the start of the range.
<i>Use Minimum</i>	If this parameter is set to <i>off</i> , the start of the range is considered to be negative infinity.
<i>Maximum</i>	The following parameter allows defining the end of the range.
<i>Use Maximum</i>	If this parameter is set to <i>off</i> , the end of the range is considered to be infinity.
<i>Is Value In Range Datum</i>	The following parameter allows defining the datum that will be used store whether or not the value is between <i>Minimum</i> and <i>Maximum</i> .

**[00253]** Active Animation Sensor

**[00254]** An active animation sensor can set the value of a datum based on whether or not a certain animation is active.

**[00255]** The following tables includes examples of parameters that can be used to define a state sensor:

Parameter	Description
<i>Animation</i>	This parameter allows defining the animation to be sensed.

Parameter	Description
<i>Is Animation Active Datum</i>	This parameter allows defining the datum that will be used to store whether or not the animation is active.

**[00256]**            Commands, Decisions, and Decision Trees

**[00257]**            As illustrated in steps 110-114 of Figure 1, decision trees are used to process the data information gathered using sensors.

**[00258]**            Step 110 results in a Command being used to activate a behaviour or an animation, or to modify an AIE's internal memory.

**[00259]**            Commands are invoked by decisions. A single Decision consists of a conditional expression and a list of commands to invoke.

**[00260]**            A Decision Tree consists of a root decision node, which can own child decision nodes. Each of those children may in turn own children of their own, each of which may own more children, etc.

**[00261]**            Figure 5 illustrates a method of use of a Decision Tree to drive Action Selection. The method of Figure 5 corresponds to step 110 on Figure 1.

**[00262]**            Since the method 110 iterates on all frames, verification is done in step 118 to verify whether all frames have been processed. A similar verification is done in step 120 for the AIEs.

**[00263]**            In step 122, all of the current AIE's behaviours are deactivated.

**[00264]** In step 124, verification is done to insure that all decision trees have been processed for the current AIE.

**[00265]** Then, for each decision tree, the root decision node is evaluated (step 126), all commands in the corresponding decision are invoked (step 128), and the conditional of the current decision tree is evaluated (step 130).

**[00266]** It is then verified in step 132, whether all decision nodes have been processed. If yes, the method 110 proceeds with the next decision tree (step 124). If no, the child decision node indicated by the conditional is evaluated (step 134), and the method returns to the next child decision node (step 132).

**[00267]** For the example given hereinabove, where a character moves along a path while running away from any nearby enemies, the following elements can be created:

- two behaviours: FleeFromEnemy and FollowPath;
- a datum called IsEnemySeen for the character to store whether or not it sees the enemy;
- a vision sensor that looks for the enemy. This would feed into the Is EnemySeen datum; and
- the following decision tree:

```

if (IsEnemySeen)
|
|—then: activate FleeFromEnemy
|
|—else: activate FollowPath

```

**[00268]** It results from the above decision tree that when the character sees the enemy, it will activate its Flee From behaviour and if the character does not see the enemy it will activate its Follow Path behaviour.

**[00269]** Note that if an AIE is assigned a decision tree, the solver deactivates all behaviours before solving for that AIE (step 122 in Figure 5). In this last example, at every frame, both the “FleeFromEnemy” and “FollowPath” behaviours are deactivated. Then, based on the value of the “IsEnemySeen” datum, one of them is reactivated.

**[00270]** A parameter indicative of whether or not the decision tree is to be evaluated can be used in defining the decision tree.

**[00271]** Whenever the command corresponds to activating an animation and a transition is defined between the current animation and the new one, then that transition is first activated.

**[00272]** Similarly, whenever the command corresponds to activating a behaviour, a blend time can be provided between the current animation and the new one.

**[00273]** Moreover, whenever the command corresponds to activating a behaviour, the target is changed to the object specified by a datum. For example, to make a character flee from the nearest enemy:

- a group called “enemies” can be created to include all the

ennemies;

- a “Flee From” behaviour is created for the character called “FleeFromNearestEnemy”. At this point the target of the behaviour is not yet defined;
- a datum is added to the character, called “NearestEnemy”;
- another datum is assigned to the character, called “IsAnyEnemySeen”;
- a vision sensor is created and assigned to the character called “EnemySensor”. This cause the “enemies” to group as an object filter, “distance” as the evaluation function, “minimum” as the value for the Min Max attribute (if “maximum” is chosen, then the seen enemy within the visibility distance will be chosen). Also, the “Is Any Object Seen” datum is set to “IsAnyEnemySeen” and the “Best Object” datum to “Nearest Enemy”. If an enemy is within the sensor’s visibility distance “IsAnyEnemySeen” will be set to true and “NearestEnemy” will contain a reference to the nearest enemy, otherwise if no enemy is seen, “IsAnyEnemySeen” will be set to false;
- a DecisionTree is created as follows:

```

if (IsAnyEnemySeen)
|
|—then: FleeFromEnemy
|
|—else: DoNothing

```

- the FleeFromEnemy decision results in a Change Behaviour Target Command with "FleeFromNearestEnemy" as the behaviour and "NearestEnemy" as the datum (meaning change the target of the "FleeFromNearestEnemy" behaviour to the value of the "NearestEnemy" datum).

**[00274]** Examples of command that can be used with the method 100, and more specifically in step 112 or 114, will be described.

**[00275]** Queue Animation Command

**[00276]** This command can be used to activate an animation (the "New Animation") once another one (the "Old Animation") completes its current cycle. For example, a queue animation with a walk animation as the "Old Animation", and a run animation as "New Animation", will activate the run animation as soon as the walk animation finishes its current cycle.

**[00277]** It is to be noted that the same result can be achieved by defining a Queuing transition between the animations, then using a normal ActivateAnimation command.

**[00278]** Set Datum Command

**[00279]** This command can be used to set (or increment, or decrement) the value of a datum. If the datum represents an AIE's state, then this command can be used to transition between states.

**[00280]** Set Value Command

**[00281]** This command allows setting (or increment, or decrement) the value of any attributes of one or more AIEs or character characteristics (such as behaviours, sensors, animations, etc).

**[00282]** In particular, this command may be used to set a character's position and orientation, active state, turning radii, surface offset, etc.

**[00283]** The set value command may include two parts: the Item Selection part for specifying which items are to be modified, and the Value section for specifying which of the item's attributes are to be modified.

**[00284]** Group Membership Command

**[00285]** This command can be used to add (or remove) an AIE to (from) a group. Also, such a command may be used to remove all members from a group.

**[00286]** Animation Control

**[00287]** It will now be described how animation clips are associated to AIEs and how an AIE drives the right animation clip at the right time and speed.

**[00288]** Animation clips can be defined for example using one of the following parameters:

Parameter	Description
<i>Active</i>	This parameter allows defining whether or not the animation is active
<i>Length</i>	This parameter allows defining the number of frames the animation will take to perform a full cycle. Normally, this number would correspond to the length of the clip itself. If it doesn't, then the animation will be scaled to fit the length indicated by this attribute.

Parameter	Description
<i>Speed Adjusted</i>	This parameter allows defining whether or not the animation depends on the speed of the AIE. If so, the faster the AIE moves, the faster the animation is played. For example, a walk clip should be speed adjusted but an idle clip should not.
<i>Preferred Speed</i>	If <i>Speed Adjusted</i> is set to <i>off</i> , this parameter is ignored. Otherwise, it defines the speed of an AIE at which the animation will take as many frames as defined by the <i>Length</i> attribute to perform a full cycle. For example, if a walk clip has a length of 10 and a preferred speed of 3, then the walk clip will take 10 frames to perform a full cycle when the AIE moves at a speed of 3. However, if the AIE moves at a speed of 6 it will take only 5 frames.
<i>Cyclic</i>	This parameter allows defining whether or not the animation clip will repeat itself. A non-cyclic animation will only perform one cycle when it is activated.
<i>Interruptible</i>	If this parameter is false, then no other animation may be activated for this AIE until this animation finishes. Default value is true.

#### [00289] Animation Selection

[00290] Action Selection allows, for example, to switch between an idle animation and a walk animation based on its speed. In this particular case the first step is to create a datum for the character, which might be called "IsWalking". Next, a speed sensor is created to drive the datum. Then the following decision tree is created to activate the correct animation depending on the speed of the character:

```

if (IsWalking)
|
|—then: activate walkAnimation
|
|—else: activate idleAnimation

```



**[00291]**            Thus, at each frame, either the walk or idle clip would be played based on the value of the IsWalking datum.

**[00292]**            Action Selection effectively allows activating the right animation at the right time, and adjusting its scale and number of cycles appropriately.

**[00293]**            Animation Transitions and Blending

**[00294]**            While animation commands are used to activate or deactivate clip animations at the appropriate time, animation transitions are used in order to specify what happens if an animation is already playing when another one is activated. This allows creating a smooth transition from one animation to another. In particular, animation transitions make it possible to smoothly blend one clip into another.

**[00295]**            Before describing in more detail the action selection and the use of transitions between the activation of animation clips, the following terminology is introduced:

**[00296]**            Animation channel: attributes of objects with animation curves;

**[00297]**            Animation clip: a list of channels and their associated animations. Typically the animation of each channel is defined with a function curve, that is, a curve specifying the value of the parameter over time. This concept promotes animation reuse, over several characters and over time for a given AIE. It does this by scaling, stretching, offsetting the animation and possibly fitting it for a specific AIE. Common examples for a clip are walk or run cycles, death animations, etc.

**[00298]** Animation blending: a process that computes the value for a channel by averaging two or more different clips, in order to get something in-between. This is generally controlled by a weight that specifies the amount of each animation to use in the mix.

**[00299]** Interpolation/Transitions: a blending that occurs from either a static, non-animated posture to a new pose or animation, or an old animation to a new one where the new animation is expected to take over completely over a transition time.

**[00300]** Marker: used to define a reference point in an animation clip for transitions. For example in the last frame of the “in” animation clip a character has their right foot on the ground, the marker could then be used to define a similar position in the “out” animation to transition to.



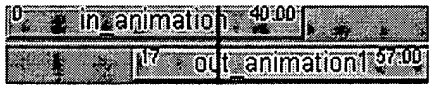
**[00301]** Animation markers are reference points allowing synchronizing clips. They are used to synchronize transitions between two clips.

**[00302]** The following table includes example of parameters that can be used to define animation markers:

Parameter	Description
<i>Available Markers</i>	This parameter allows defining markers that currently available to be used.
<i>Markers used in this animation</i>	This parameter allows defining markers that are used within the current animation. When selected all corresponding animations that utilize that same animation marker can be displayed to the user so as to help setting animation.

**[00303]** The following table includes example of parameters allowing to define animation blending:

Parameter	Description
<i>Start Marker</i>	This parameter allows defining which marker will be used to start the new animation. If no marker is specified then the beginning of the animation will be used.
<i>Blending</i>	This parameter allows defining a period of time over which both animations will be playing simultaneously, the old progressively morphing into the new. The following two parameters allows controlling the blend into the incoming animation:
<i>BlendDuration</i>	The length of the blend in frames.
<i>Blend Type</i>	Linear or smooth. Linear is a constant progression while a smooth blend is an ease-in/ease-out transition.

Parameter	Description
<i>Transition Type</i>	<p>The type "Interrupt" allows for an immediate transition to a new animation as illustrated in the following example.</p>  <p>The type "Queue" allows for a transition that occurs only when the current cycle of animation is complete, as illustrated in the following example.</p>  <p>The type "AutoSynchronize" allows for a transition using common markers between animations. The current animation will continue to play until it reaches the next common marker and only then will the transition occur. For example, in the in_animation, a marker is created at frame 30. The same marker is used for animation clip 2 at frame 25. The resulting transition will always occur at frame 30 for the in clip and 25 for the out clip, as illustrated in the following example.</p>  <p>When transitioning between cycled clips, AutoSynchronize will attempt to dynamically choose the most appropriate transition point for the incoming clip based on the position of a common marker.</p>
<i>In-between animation</i>	<p>This parameter allows defining the option of playing another, third animation before moving on to the new animation. If an in-between animation is used, there will be additional blending parameters.</p>

Parameter	Description
<i>In-between blend</i>	<p>If one uses an in-between animation, then there are in fact 2 transitions that will occur: one between the old animation and the in-between animation, and another between the in-between animation and the new. The parameters used for the latter are the ones we saw previously. For the former one, an in-between blend duration and a blend type are specified.</p> <p>* If there is no transition defined between two animations, then for the moment a transition type of interrupt is used, and a blend time as controlled by the character attribute "Default Animation Blend Time" (whose default value is zero).</p>

**[00304]** It is to be noted that character's attributes can also be used to define animation. The following table includes examples of such character's attributes:

Parameter	Description
<i>Default Animation Blend Time</i>	If no transition is defined between two animations the default Animation Blend Time allows creating an interrupt transition of the specified length.
<i>First Animation Frame</i>	This parameter allows specifying the initial frame of animation played for the first active animation of an AIE.

**[00305]** According to a further aspect of the present invention, waypoints are provided for marking spherical region of space in the virtual world.

**[00306]** Characteristics and functions of waypoints will be described with reference to the following table, including examples of parameters that can be used to define waypoints:

Parameter	Description
<i>Exists</i>	This parameter allows defining whether or not the waypoint exists in the solver world. If this is set to <i>off</i> the solver ignores the waypoint.
<i>Collidable</i>	This parameter allows defining whether or not collisions with other collidable objects will be resolved.
<i>Radius</i>	This parameter allows defining the radius of the waypoint's bounding sphere. When an AIE follows a path, if it is inside the bounding sphere of a waypoint the AIE will seek to the next waypoint.
<i>Speed Limit</i>	This parameter is only used for Paths, as will be described hereinbelow. It indicates the desired speed an AIE will have when approaching this waypoint when following the path. This speed limit will only be heeded if the <i>Use Speed Limits</i> attribute of the Follow Path behaviour has been set to <i>on</i> .
<i>IsPortal</i>	This parameter is only used for Waypoint Networks, which will be described hereinbelow. It informs the solver that this waypoints stands at the doorway to a room. The solver uses this information to optimize performance for large networks.

**[00307]** Waypoints allow to creates a path, which is an ordered set of waypoints that AIE may be instructed to follow. For example, a path around a racetrack would consist of waypoints at the turns of the track.

**[00308]** Each waypoint can be assigned speed limits to control how the AIE approaches it (e.g. approach this waypoint at this speed). Paths can be used to build racetracks, attack routes, flight paths, etc.

**[00309]** Also, linking together waypoints with edges may create a waypoint network. For example, a character with a *SeekToViaNetwork* behaviour can use a waypoint network to navigate around the world. An edge between a waypoint in the network to another waypoint in the same network indicates that an AIE can travel from the first waypoint to the second. The lack of an edge between two waypoints indicates that an AIE cannot travel between them.

**[00310]** A waypoint network functions in a similar manner to a path however, the exact route that the autonomous character takes is not pre-defined as the character will navigate its way via the network according to its environment. Essentially a waypoint network can be thought of as a dynamically generated path. Collision detection is used to ensure that AIEs do not penetrate each other or their surrounding environment.

**[00311]** According to a further specific aspect of the present invention, there is provided a method for generating waypoints network. The method includes analyzing the level, determining all reachable areas and placing the minimum necessary waypoints for maximum reach ability. For example, reachable waypoints can be positioned within the perimeter of the selected barriers, and outside barrier enclosed unreachable areas or “holes”.

**[00312]** It is to be noted that a waypoint can be positioned in the virtual world at the entrance to each room and mark is as a portal using a specific waypoint's parameter. Each room can have one Portal waypoint per doorway. Thus any 2 rooms connected by a doorway will have 2 Portal waypoints (one just inside each room) connected by an edge, and all passages or doorways connecting one room to another will have a corresponding edge between 2 Portal waypoints. All other waypoints should have the “IsPortal” parameter set to off. This allows the solver to significantly reduce the amount of run-time memory required to navigate large networks (i.e. > 100 waypoints).

**[00313]** Figure 6 illustrates a system 200 for on-screen animation of digital entities according to a first illustrative embodiment of a second aspect of the present invention. The system 200 is in the form of a computer application plug-in 204 embodying the method 100 and inserted into the pipeline and workflow of an existing computer animation package (or platform) 202, such as Maya™ by Alias Systems, and 3ds-max™ from Discreet. Alternatively, it can also be implemented as a stand-alone application.

**[00314]** The animation package 202 includes means to model and texture characters 206, means for creating animation cycles 208, means to add AI animation to characters 210, and means to render out animation 212. Since those last four means are believed to be well known in the art, and of concision purposes, they will not be described herein in more detail.

**[00315]** The plug-in 204 includes an autonomous entity engine (AEE) (not shown), which calculates and updates the position and orientation of each AIE for each frame, chooses the correct set of animation cycles, and enables the correct simulation logic.

**[00316]** The plug-in 204 is designed to be integrated directly into the host art package 202, allowing animators to continue animating their AIE via the package 202, rather than having to learn a new technology. In addition, many animators are already familiar with specific animation package workflow, so learning curves are reduced and they can mix and match functionality between the package 202 and the plug-in 204 as appropriate for their project.

**[00317]** The system 200 may include a user-interface tool (not shown) for displaying specific objects and AIEs from the digital world, for example, in a tree view structure orientated from an artificial perspective. This tool allows selecting multiple objects and AIEs for editing one or more of their attributes.

**[00318]** Furthermore, a Paint tool (not shown) can be provided to organize, create, position and modify simultaneously a plurality of AIEs. The following table includes examples of parameters that can be used to define the effect of the paint tool:

Parameter	Description
<i>Group Name:</i>	This parameter allows specifying the name of the current group or paint layer.



Parameter	Description
<i>Group:</i>	This parameter allows specifying the currently active group.
<i>Individuals:</i>	This parameter allows specifying the name for a subgroup or paint layer.
<i>Individual names:</i>	This parameter allows specifying the currently active Individual group.
<i>Variation:</i>	Variations add a version number to the full name of the proxy character. i.e. Warrior_Var10_Roman_Grp1. This then would read as the Warrior of type 10 that is member of the Roman group.
<i>Assign Vertex Color:</i>	This parameter allows applying the selected vertex color to the active layer.
<i>Proxy: Create/Modify</i>	This parameter allows creating new proxy characters or modifies existing AIEs based on the active Control options.
<i>Proxy: Modify</i>	This parameter allows modifying proxy characters based active Control options. This is useful to perturb the position orientation and scale of an AIE.
<i>Proxy: Remove</i>	This parameter allows removing the proxy AIEs.
<i>Selection: Select Deselect Toggle</i>	This parameter allows selecting deselecting or toggling the current selection.
<i>Modify:/Create:</i>	This parameter allows selecting the type of object to modify or create.
<i>Paint Attribute:</i>	This parameter allows selecting the attribute to paint the value of.
<i>Grid</i>	This parameter allows enabling objects to be painted at positions other than the vertices.
<i>Jitter Grid</i>	This parameter allows randomizing the placement of objects.
<i>U V Grid Size:</i>	This parameter allows defining the density of painted objects.
<i>Control:</i>	This parameter allows controlling options used to specify which transform attributes are to be modified.
<i>Options: Group</i>	This parameter allows parenting Proxy characters to a group node.
<i>Options: Align</i>	This parameter allows aligning proxy characters to the normal of the surface.

Parameter	Description
<i>Jitter Value:</i>	This parameter allows defining a percentage of randomness applied to the Jitter Grid.
<i>Vertex Color Display</i>	This parameter allows enabling the vertex color display for the active layer.

**[00319]** The system 200 for on-screen animation according to the present invention provides for means to duplicate the attributes from a first AIE to a second AIE. The attributes duplication means may include a user-interface allowing selecting the previously created recipient of the attributes, and the AIE from which the attributes are to be copied. Of course, the duplication may extend also to behaviours, animation clips, decision tree, sensors and to any information associated to an AIE. The duplication allows to simply and rapidly create a group of identical AIE.

**[00320]** More specifically, options are provided to precise the attribute duplication process, such as:

Option	Description
<i>Copy Key Frames</i>	This option defines whether or not any key frames and driven keys on the source AIE should be duplicated. If this option is not selected, any attributes controlled by key frames (or are driven keys) have only their values duplicated.
<i>Copy Expressions</i>	This option defines whether or not any expressions on the source AIE should be duplicated. If this option is not selected, any attributes controlled by expressions only have their values duplicated.
<i>Tag Proxy</i>	This option defines whether or not the proxy AIE should have connection information written to later reconnect animations.
<i>Copy Behaviours</i>	This option defines whether or not the behaviours of the source AIE should be duplicated.
<i>Copy Animations</i>	This option allows defining whether or not the animations associated to the source AIE should be duplicated.

Option	Description
<i>Copy Action Selection</i>	This option defines whether or not the action selection components of the source AIE should be duplicated. This includes data, sensors, decision trees, decisions, and commands.
<i>Copy Groups</i>	This option defines whether or not the destination AIEs should be put in the same group or groups as the source autonomous character.
<i>Remove Autonomous Characters</i>	This option defines whether or not the AIEs (if any) of the destination objects should be removed before duplication of the source object. If this is selected then all the Artificial Intelligence (AI) specific information of the destination AIE, except for its name, is removed before duplication. If this is not set, then the components of the source AIE are added to those of the destination AIEs.
<i>Remove Behaviours</i>	This option defines whether or not the behaviours of the destination AIEs should be removed before duplication. If this option is not selected, then the behaviours of the source AIE are added to those of the destination AIEs.
<i>Remove Animations</i>	This option defines whether or not the animations of the destination AIEs should be removed before duplication. If this option is not selected, then the animations of the source AIE are added to those of the destination AIEs.
<i>Remove Action Selection</i>	This option defines whether or not the action selection components of the destination AIEs should be removed before duplication. If this is not selected, then the action selection components of the source AIE are added to those of the destination AIEs. Action selection components include data, sensors, decision trees, decisions, and commands.
<i>Remove Groups</i>	This option defines whether or not the destination AIEs should be removed from their groups before duplication. If this option is not ticked, then the groups of the source AIE are added to the destination AIEs.

**[00321]** Alternatively or additionally, the duplication process may be performed on an attribute-to-attribute basis.

**[00322]** Turning now to Figures 7 and 8, the method 100 will now be described by way of a first specific example of application related to the

animation of a school of fish 302 and a hungry shark 304 in a large aquarium 306. Figure 7 illustrates a still image 300 from the computer animation.

**[00323]** The walls of the aquarium are defined as barriers 308 in the virtual world, the seaweed 310 as non-autonomous image entities, and each fish 312 and shark 304 as autonomous image entities.

**[00324]** Each fish 312 is assigned a “Flock With” other fish behaviour so that they all swim in a school formation, as well as a “Wander Around” behaviour so that the school 302 moves around the aquarium 300. To allow a fish 312 to escape the hungry shark 304, it is assigned the behaviour “Flee From” shark. The “Flee From” behaviour is given an activation radius so that when the shark 304 is outside this radius the “Flee From” behaviour would effectively be disabled and only enabled when the shark 304 is inside the radius.

**[00325]** To prevent a fish 312 from hitting other fish 312, the seaweed 310, nor the aquarium walls 308, each fish 312 has the additional behaviours “Avoid Obstacles” (seaweed 310 and the other fish 312) and “Avoid Barriers” (the aquarium walls 308). Similarly to the case in real life, the solver resolves these different behaviours to determine the correct motion path so that, in its efforts to avoid being eaten, a fish 312 avoids the shark 304, the other fish 312 around it, the seaweed 310, and the aquarium walls 308 the best it can.

**[00326]** Considering the case when a fish 312 wants to escape the hungry shark 304. At this point in time, both a fish’s “Flee From” shark and “Flock With” other fish behaviours will be activated causing two steering forces to act on the fish 312 in unison. Therefore, a fish 312 will try to escape the shark 304 and stay with the other fish 312 at the same time. The resulting active steering force on the fish 312 will be the weighted sum of the individual behavioural forces, based on their intensities. For example, for the fish 312, it is

much more important to flee from the shark 304 than to stay in a school formation 302. Therefore, a higher intensity is assigned to the fish's "Flee From" behaviour than the "Flock With" behaviour. This allows the fish 312 to break formation when trying to escape the shark 304 and then to regroup with the other fish 312 once it is far enough away from the shark 304.

**[00327]** Although simply adjusting the fish's behaviours intensities allow yielding realism, alternatively the "Flock With" behaviour of the fish 312 can be disabled and its "Flee From" behaviour is enable when the fish 312 sees the shark 304. Once out of range of the shark 304, a fish 312 would then continue to swim in a school 302 by disabling its "Flee From" behaviour and enabling its "Flock With" behaviour. This type of behavioural control can be achieved by setting the behaviours' priorities. By giving the "Flee From" behaviour a higher priority than the "Flock With" behaviour, when a fish 312 is fleeing from a shark 304 its "Flock With" behaviour will be effectively disabled. Assigning such priorities to the behaviours causes a fish 312 not to try remaining with the other fish 312, while trying to flee the shark 304. However, once it has escaped the shark 304 the "Flock With" behaviour is reactivated and the fish 312 regroups with its school 302.

**[00328]** In many relatively simple cases such as this one, to obtain a realistic animation sequence, it is usually sufficient to assign various degrees of intensities and priorities to specific behaviours. However, in a more complicated scenario, simply tweaking a behaviour's attribute may not produce acceptable results. In order to implement higher-level behaviour, an AIE needs to be able to make decisions about what actions to take according to its surrounding environment. According to the method 100, this is implemented via Action Selection.

**[00329]** The steering behaviour mechanisms described above allows controlling the behaviour of AIEs. However, an AIE often warrants greater

intelligence. A method and system according to the present invention enables an animator to assign further behavioural detail to a character via Action Selection. Action Selection allows AIEs to make decisions for themselves based on their environment, where these decisions can modify the character's behaviour, drive its animation cycles, or update the character's memory. This allows the animator to control which behaviours or animation cycles are applied to an autonomous character and when.

**[00330]** Alternatively to assigning priorities to certain behaviours, a vision sensor is created for each autonomous fish 312 to determine whether the fish 312 sees a shark 304 or not.

**[00331]** Figure 8 illustrates a decision tree 320 created and used to implement Action Selection for the fish 312. Therefore, during each think cycle, a vision sensor created for each fish 312 produces a datum true or false in response to the question; Do I see a shark? 322. A set of rules is then created for the AIE (the fish 312) to apply to the data to be gathered from the vision sensor. For instance, if a fish 312 sees a shark 304 then it should swim away from the shark 304 (step 324) and the "Flee From" shark behaviour is activated. If not, then the fish 312 should flock with any similar fish 312 within thirty centimeters (step 328) and its "Flock With" other fish behaviour is activated (step 330). To further enhance the simulation, other decision trees could be used to activate and control animation clips as well as simulation logic.

**[00332]** The method 100 will now be described by way of a second specific example of application related to the animation of characters in a battle scene with reference to Figures 9-14.

**[00333]** Film battles typically involve two opposing armies who run towards each other, engage in battle, and then fight until one army weakens and is defeated. Given the complexity, expense, and danger of live filming such

scenes, it is clear that an effective AI animation solution is preferable to staging and filming such battles with actual human actors.

**[00334]** The present example of application of the method 100 involves an army 401 of 250 disciplined Roman soldiers 403, composed of 10 units led by 10 leaders, against a horde 405 of 250 beast warriors 407 composed of 10 tribes lead by 10 chieftains. The scenario is as follows. The Romans 403 are disciplined soldiers who marched slowly in formation until the enemy is very close. Once within fighting range, the Romans 403 break ranks and attack the nearest beast warrior 407 (see for example Figure 11C). The Romans 403 never retreat and will fight until either themselves or their enemy have all been killed. The beast warriors' tactics are completely opposite to the Romans'. The beast warrior chieftains run at full speed towards the Roman army 401 and attack the closest soldier 403 they find. Individual beast warriors 407 follow their chieftain and fight the Romans 403 as long as their chieftain is alive. Once their chieftain is killed, they retreat.

**[00335]** The following description outlines how the method 100 can be used to animate this battle scene. Firstly, group behaviour and the binary decision tree that determines what actions the characters 403 and 407 will make are defined. Secondly, individual character behaviour and the binary decision tree to ensure that the correct animation cycle is played at the correct time are defined.

**[00336]** In a battle involving hand-to-hand combat (see Figure 13A-13C), there are typically two different ways for enemy groups to engage one another: marching in a tight, often geometric, formation or running together in a basic horde. Being disciplined soldiers the Romans 401 choose the first manner, while the beast warrior tribes 405 the second.

**[00337]** The Roman soldiers 403 and their leaders behave in exactly the same manner. As summarized in Figure 9, they are made to march in formation by initially laying them out in the correct geometric formation and then applying a simple “Maintain Speed At” behaviour to start them marching and an “Orient To” beast warriors behaviour to point them in the right direction. Once the soldiers 403 are sufficiently close to the opposing beast warriors 407, these behaviours are de-activated by their binary decision trees and replaced by their tactical behaviours, as illustrated with the decision tree 400 of Figure 9. In order to deactivate and activate the soldiers’ behaviours, sensors are used to determine specific datum points. For example, to determine if a soldier 403 sees a beast warrior we create a vision sensor to answer the question “Do I see a beast warrior?” (step 402). To this question the sensor will return either true or false. Based on this response, the soldier 403 will decide how to act. If the soldier 403 sees a warrior 407, he will determine if the warrior 407 is within fighting distance (step 404). If so, he will attack the warrior 407 (step 406), if not he will “Seek To” the warrior (step 408). If the soldier 403 does not see a warrior 407, he will continue marching towards beast warriors 407 (step 410).

**[00338]** In contrast to the Romans 403, the beast warriors 407 run towards their enemy as a pack. The beast warrior chieftains are made to run towards the Romans 401 by setting their behaviour as “Seek To” the group of Romans at maximum speed. The beast warriors 407 in turn follow their chieftains via a “Seek To” chieftain behaviour. Once the beast warriors 407 are within range of the Roman soldiers 403, these behaviours are de-activated by their binary decision trees and replaced by their tactical behaviours, in much the same manner as we previously did for the Roman soldiers. The tactical behaviour binary decision tree 412 for a beast warrior 407 is illustrated in Figure 10. If a beast warrior 407 sees his chieftain, (step 414) i.e. the chieftain is still alive, he will fight with any Roman soldier 403 (step 418) who is within fighting distance (step 416), or “Seek To” the closest soldier if no soldier is



nearby (420). If a chieftain is killed, the beast warriors 407 of his tribe will run away from the surrounding Romans with a "Flee From" group of Romans behaviour (Step 422). The fight behaviour of the chieftains is the same as for their warriors 407, except that obviously they do not first determine if they see their chieftain before determining if a Roman soldier is within fighting distance.

**[00339]** The binary decision trees illustrated in Figures 9 and 10 generally dictate how the 250 Roman soldiers and the 250 beast warriors engage in battle. When the animation is run, the battle would typically proceed as shown in Figures 11A-11D. These screen shots are taken from a complete battle animation.

**[00340]** Once the gross motion of the battle is complete, the close-up hand-to-hand combat remains to be animated (see Figures 13A-13C and 14). Although each character is very small in the final shot, it is important for special effects artists to have the most realistic scene possible.

**[00341]** The decision trees illustrated in Figures 9 and 10 end as the character is about to fight an enemy character. As the real battle doesn't stop there the manner in which each character engages in hand-to-hand combat is to be determined. The Romans 403 and beast warriors 407 according to the present example fight in similar fashions. Once a character is within striking range of its target, the enemy, the binary decision tree for the fight sequence randomly chooses between an upper and lower weapon attack. If the attack is unsuccessful the character keeps fighting until it either kills its enemy or is killed itself. If the attack is successful, then the target character plays its dying animation sequence that corresponds to how it was attacked. For example, if a character was killed via an upper weapon attack it will play its upwards dying sequence, and if the attack was a lower weapon attack, it will play its downwards dying sequence.

**[00342]** In order to play the correct animation sequence for each character during each fight sequence, the binary decision tree 424 shown in Figure 12 is implemented. This decision tree 424 determines which animation clip to play and at what time according to the actions that the character performs. The decision tree 424 is created in a similar manner as the behaviour decision trees previously discussed, in that datum points are created and sensors are used to determine the data. However, instead of changing the behaviour of the character, the output of the decision tree determines which animation clip to play.

**[00343]** In this example, it is first determined whether a character is walking or not via a speed sensor (step 426). The information returned from the sensor allows determining whether a walk or idle animation sequence should be played. It is then determined whether the character is attacking the enemy or not (428-428'). This information allows determining whether a fight animation is to be played. In order to choose which fight animation to play (FightHigh or FightLow), a random sensor is used to randomly return true or false each cycle. As the binary decision tree does not guarantee that a FightHigh animation sequence will be completed before a FightLow sequence is played, or vice versa, a given animation sequence is queued if another one is currently active. This ensures that a given fight animation sequence is completed before the next sequence commences. As each type of character has different animation sequences, the decision tree 424 is duplicated for each type of character and the correct animation sequences are associated thereto.

**[00344]** The animation sequence resulting from the second example can be completed by creating a decision tree for the dying sequence of each character. Then the required number of characters necessary to fill the battleground is duplicated and the animation is triggered. The screen shots shown in Figures 13a-13c and in Figure 14 are taken from the battle animation according to the second example.

**[00345]** The method 100 will now be described in more detail with reference to other specific examples of applications related to the animation of entities.

**[00346]** An animation logic wherein two humans walk through a narrow corridor cluttered with crates that they must avoid will now be considered, the elements defining the scene being:

**[00347]** two humans, defined as AIE;

**[00348]** an animation cycle associated to each humans to drive their walking animation;

**[00349]** a path stretching from one end of the corridor to the other;

**[00350]** the crates being non-autonomous entities;

**[00351]** the two humans being initially assigned the behaviour "Follow Path" so that they follow the path, and the behaviour "Avoid Obstacles" in order to avoid the crates. Since the crates are non-autonomous entities, they can move around in real-time resulting in the characters adjusting their position. Optionally, the motion of the crates can be driven with another system, such as rigid-body dynamics; and

**[00352]** the walls of the corridors are defined as barriers; the two humans further being assigned an "Avoid Barriers" behaviour so that they do not walk into the walls.

**[00353]** Another example includes characters moving as a group. Group behaviours enable grouping individual autonomous characters so that they act as a group while still maintaining individuality. According to this

example, a group of soldiers are about to launch an attack on their enemy in open terrain.

**[00354]** The soldiers are defined as AIEs and any obstacles, such as trees and boulders, are defined as non-autonomous entities.

**[00355]** As the ground is not perfectly flat, a flat terrain is created and the height fields of various points are modified to give the terrain some elevation. To ensure that the soldiers remain on the ground it is provided that they hug the terrain.

**[00356]** To prevent the soldiers from walking into obstacles each soldier is assigned an “Avoid Obstacles” behaviour

**[00357]** To ensure that the soldiers remain as a unit they are also assigned a “Flock With” behaviour that would specify how closely they keep together.

**[00358]** A “Seek To” the enemy behaviour is finally assigned to make the soldiers move towards their enemy.

**[00359]** According to a further example, there is provided a car race between several cars that occurs on a racetrack.

**[00360]** The cars are defined as AIEs. Each car is defined by specifying different engine parameters (max. acceleration, max. speed, etc.) so that they each race slightly differently.

**[00361]** As the racetrack is not perfectly flat, a flat terrain is first created within the digital world and then the height fields are changed at

various points to give the terrain some elevation. To ensure that the cars stay on the surface of the track, it is specified that the cars hug the terrain.

**[00362]** A looped path that follows the track is provided and the cars are assigned a "Follow Path" behaviour so that they stay on the racetrack. Each waypoint along the path is characterized by a speed limit associated to it (analogous to real gears at turns) that would limit the speed at which a car could approach the waypoint.

**[00363]** To prevent the cars from crashing into each other, each car is further characterized by an "Avoid Obstacles" behaviour as each car can be considered an obstacle to the other cars.

**[00364]** Finally, in order to keep the cars from straying too far off the racetrack, hidden barriers are added along the sides of the track and an "Avoid Barriers" behaviour is assigned to each car.

**[00365]** The next example concerns a skateboarder in a skate park.

**[00366]** The skateboarder is defined as an AIE and the various obstacles within the park, such as boxes and garbage bins, as obstacles. The ramps upon which the skateboarder can skate are defined as surfaces and to ensure that the skateboarder remains on a ramp surface rather than pass through it, it is specified that he hug the surface.

**[00367]** As discussed hereinabove, for AIEs to be able to make decisions for themselves based on information about their surrounding environment, Action Selection is implemented. For example, a guard patrolling a fortified compound against intruders is now provided as an example of animation according to the method 100.

**[00368]** The guard is defined as an AIE, the buildings and perimeter fence as barriers, and the trees, vehicles etc. within the compound as non-autonomous entities. A flat terrain is first created and then the height fields of various points are modified to give the terrain some elevation. To ensure that the guard remains on the ground it is specified that he hugs the terrain.

**[00369]** To prevent the guard from walking into obstacles within the compound during his patrol, an "Avoid Obstacles" behaviour is assigned thereto. In addition, to prevent him from walking into the perimeter fence or any of the buildings, he is also assigned an "Avoid Barriers" behaviour.

**[00370]** To specify the route that the guard takes during his patrol, a waypoint network is provided and the guard is assigned a "Seek To Via Network" behaviour. A waypoint network rather than a path is used to prevent the guard from following the exact same path each time. Via the network, the guard dynamically navigates his way around the compound according to the surrounding environment.

**[00371]** Sensors are created allowing the guard to gather data about his surrounding environment and binary decision trees are used to decide what actions to take to enable the guard to make decisions about what actions to take. For instance, sensors are created to enable the guard to hear and see in his surrounding environment. If he heard or saw something suspicious he would then decide what to do via a binary decision tree. For example, if he heard something suspicious during his patrol, he moves towards the source of the sound to investigate. If he didn't find anything, he returns to his patrol and continue to follow the waypoint network. If he did find an intruder, he fights the intruder. Further sensors and binary decision trees can be created to enable the guard to make other pertinent decisions.

**[00372]** Figures 15 and 16 describe a system 502 for on-screen animation of digital entities according to a second illustrative embodiment of the second aspect of the present invention. This second illustrative embodiment of the second aspect of the present invention concerns on-screen animation of entities in a video game.

**[00373]** The system 502 is in the form of an AI agent engine to be included in a video game platform 500. The AI agent 502 is provided in the form of a plug-in for the video game platform 500. Alternatively, the AI agent can be made integral to the platform 500.

**[00374]** The AI agent 502 comprises programming tools for each aspect of the game development, including visual and interactive creation tools for level edition and an extensible API (Application Programming Interface).

**[00375]** More specifically, the game platform 500 includes a level editor 504 and a game engine 506. As it is well known in the art, a level editor is a computer application allowing creating and editing the "levels" of a video game. An art package (or art application software) 508 is used to create the visual look of the digital world including the environment, autonomous and non-autonomous image entities that will inhabit the digital world. Of course, an art package is also used to create the looks of digital entities in any application, including movies. Since art packages and level editors are both believed to be well known in the art, they will not be described herein in more detail.

**[00376]** As illustrated in Figure 16, the game platform 500 further includes libraries 510 allowing a game programmer to integrate the AI engine 502 into an existing game engine 506. The AI engine 502 can be either authored directly by the game programmer by calling low-level behaviours or in the level editor using game designer friendly tools whose behaviour can be pre-visualized in the level editor 504 and exported directly to the game engine 506.

**[00377]** The libraries 510 provides an open architecture that allows game programmers to extend the AI functionality such as adding their own programmed behaviours.

**[00378]** The libraries 510, including the AI agent 502, allows for the following functionality:

**[00379]** 1 - Real-time authoring tools for level editors:

**[00380]** The libraries allow creating, testing and editing character logic in art package/level editor and to be exported directly to the game engine.

**[00381]** As discussed hereinabove, the libraries can be integrated via plug-in or directly into a custom editor or game engine.

**[00382]** The implementation of the creating tools in the form of libraries allows for real-time feedback to shorten the design to production cycle.

**[00383]** 2. Multi-platform:

**[00384]** The use of libraries allows to first author animations and then to publish them across many game platforms, such as Playstation 2 <sup>TM</sup> (PS2), Xbox<sup>TM</sup>, GameCube<sup>TM</sup>, Personal Computer (PC) implementing Windows 98 <sup>TM</sup>, Windows 2000 <sup>TM</sup>, Windows XP <sup>TM</sup>, or Linux <sup>TM</sup>, etc.

**[00385]** 3. High performance:

**[00386]** The use of libraries allows minimizing central processing unit (CPU) and memory usage.

**[00387]** It allows optimizing the animation for each platform.



**[00388]** 4. Open, flexible and extendable AI architecture:

**[00389]** The modularity provided with the use of libraries allows using only the tools required to perform the animation.

**[00390]** 5. Piggyback the physics layer to avoid duplicate world mark-up and representation and gain greater performance and productivity:

**[00391]** The use of libraries allows the AI agent to use the physics layer for barriers, space partition, vision sensing, etc.

**[00392]** It also allows for less environmental mark-up, faster execution, less data, less code in the executable, etc.

**[00393]** 6. Detailed integration examples of genres (e.g., action/adventure, racing, etc.) and of other middleware solutions (e.g., Renderware™, Havok™, etc.):

**[00394]** For each genre, the plug-in 502 is used to author the example. Among the covered genres include First Person Shooter (FPS), action/adventure, racing and fishing. For each genre, examples are authored and documented. This is similar for film application, where the genres include battle scene, hand-to-hand combat, large crowd running, etc.

**[00395]** For other middleware solutions, the AI agent 502 is basically integrated therewith. For physics, it can be integrated, for example, with Havok's™ physics middleware by taking one of its demo engines, ripping out its hardwired AI agent and replacing it with the AI agent 502. For rendering middleware (GameBryo™ from NDL and Criterion's RenderWare), their software are used and simple game engines are built and the AI agent is linked into them.

**[00396]** 7. Intelligent animation control feeds the animation engine:

**[00397]** Based on character decisions, animation clip control (selection, scaling, blending) is transferred to the developer animation engine. The inputs include user defined rules, and the outputs include dynamic information from each animation frame based on AI for that frame of exactly which animation cycles to play, how they are to be blending, etc.

**[00398]** A system for on-screen animation of digital entities, including characters, according to the present invention, allows creating and animating non-player characters and opponents, camera control, and realistic people or vehicles for training systems and simulations. Camera control can be created via an intelligent invisible character equipped with a virtual hand-held camera, yielding a camera that seemingly follows the action.

**[00399]** A system for on-screen animation of digital entities according to embodiments of the present invention includes user-interface menus allowing a user selecting and assigning predetermined attributes and behaviours to an AIE.

**[00400]** Also, according to some embodiments, the system for on-screen animation of digital entities includes means for creating, editing and assigning a decision tree to an AIE.

**[00401]** Of course, many user-interface means can be used to allow copying and pasting of attributes from a graphical representation of a digital entity to another. For example, a mouse cursor and mouse buttons or a user menu can be used to identify the source and destination and to select the attribute to copy.

**[00402]** A method and system for on-screen animation of digital entities can be used to animate digital entities in a computer game, in computer animation for movies, and in computer simulation applications, such as a crowd emergency evacuation.

**[00403]** Although the present invention has been described hereinabove by way of preferred embodiments thereof, it can be modified, without departing from the spirit and nature of the subject invention as defined in the appended claims.